VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PRINCIPY TVORBY A SADA DEMONSTRAČNÍCH APLIKACÍ BĚŽÍCÍCH NAD ANDROID OS

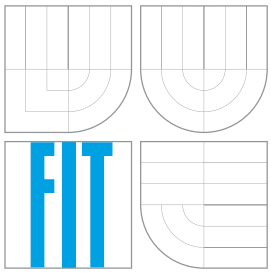
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ SZTEFEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PRINCIPY TVORBY A SADA DEMONSTRAČNÍCH APLIKACÍ BĚŽÍCÍCH NAD ANDROID OS

PRINCIPLES OF CREATION AND SET OF DEMO APPLICATIONS RUNNING OVER ANDROID OS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ SZTEFEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JOSEF STRNADEL, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Cílem práce je popsat základní principy tvorby aplikací pro operační systém Android. Čtenář se postupně seznamuje se základními stavebními prvky aplikací, včetně vytváření grafického uživatelského rozhraní. Velký důraz je věnován popisu rozhraní pro práci se senzory, na které navazuje kapitola popisující implementaci několika aplikací. Ty právě senzory využívají ke své činnosti. Po přečtení má čtenář základní povědomí o vývoji aplikací pro Android OS.

Abstract

The aim of this thesis is to describe the basic principles of creating applications for Android operating system. The reader gradually gets familiar with the basic building elements of applications, including the creation of a graphical user interface. A great emphasis is placed on the description of sensors interface, which precedes chapter describing implementation of several applications. These applications use sensors for their activity. This thesis gives a basic understanding of developing applications for Android OS.

Klíčová slova

Android OS, GUI, OpenGL ES, senzory, GPS, kompas, trasa, orientace.

Keywords

Android OS, GUI, OpenGL ES, sensors, GPS, compass, route, orientation.

Citace

Lukáš Sztefek: Principy tvorby a sada demonstračních
aplikací běžících nad Android OS, bakalářská práce, Brno, FIT VUT v Brně, 2012

Principy tvorby a sada demonstračních aplikací běžících nad Android OS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela Ph.D. Všechny zdroje, ze kterých jsem čerpal, jsem uvedl v seznamu použité literatury.

.....
Lukáš Sztefek
14. května 2012

Poděkování

Děkuji vedoucímu této práce Ing. Josefu Strnadelovi Ph.D. za ochotu přizpůsobit zadání práce mým přáním a za odbornou pomoc v průběhu řešení práce.

Obsah

Kapitola 1

Úvod

Příchod systému Android lze určitě označit jako velmi významný počin na scéně mobilních operačních systémů. Jeho zdrojové kódy jsou veřejně dostupné. Výrobci zařízení si jej mohou snadno upravovat dle potřeb svých zařízení. Aplikace jsou nejčastěji distribuovány přes integrovaný obchodní systém zvaný Google Play, což umožňuje programátorům aplikací oslovit velké množství potenciálních zákazníků. Cílem této bakalářské práce je přiblížit čtenáři hlavní stavební prvky platformy Android a základní práci s nimi.

Nejprve se zabývám operačním systémem Android obecně. Jeho účelem, zaměřením a historií. Zmiňuji se také o architektuře a celkové skladbě systému.

V další kapitole práce se soustředím na popis základních stavebních prvků aplikací, kterými jsou aktivita, služba aj. Následuje úvod do práce s různými typy úložišť. V závěru kapitoly se zabývám hierarchickým členěním Android projektů a užíváním zdrojů.

V následující kapitole uvádím metody vytváření grafického uživatelského rozhraní. Jsou zde také popsány hlavní grafické komponenty a jejich užití.

Pátá kapitola představuje možnosti systému Android v otázkách použití senzorů. Popisuji zde třídy vhodné kupříkladu pro zjišťování dostupných senzorů či čtení senzorických dat. V druhé části představuji Widget domovské obrazovky, jeho použití a proces vytváření.

V poslední kapitole demonstruji obsah předchozích kapitol na skutečných příkladech. Implementoval jsem aplikace pro zobrazení aktuálního náklonu a zrychlení v osách x/y/z, kompas s 3D střelkou a záznamník trasy s/bez použití GPS. Postup implementace je součástí právě této kapitoly. Zdrojové kódy aplikací jsou umístěny na CD přiloženém k práci.

Kapitola 2

Google Android

Android je platforma, která se v dnešní době vyskytuje na vícero druzích zařízení, jako jsou domácí multimediální centra, televize, tablety, ale hlavně, a to v drtivé většině, mobilní telefony tzv. smartphony[1]. Tato platforma se těší čím dál větší oblibě a každý den je aktivováno více než půl milionu nových Android zařízení [9].

2.1 Android historie

Společnost Android Inc. byla založena v říjnu 2005. Do akvizice společností Google se jednalo o celkem neznámou společnost, která produkovala software pro mobilní zařízení. Oficiální představení platformy Android se uskutečnilo 5. listopadu 2007 [14]. Společně s touto akcí byla představena Open Handset Alliance (dále jen OHA), která v dnešní době čítá 84 společností [10] a zaštiťuje celý Android projekt. OHA seskupuje společnosti z různých odvětví, jako jsou mobilní operátoři, softwarové firmy apod.

První Android telefon, T-Mobile G1, zvaný také HTC Dream, byl představen o necelý rok později, v září 2008. Společně s ním bylo uvedeno Android SDK[2] a o týden později spuštěn Android Market[3], nyní nazývaný Google Play. Dalším důležitým milníkem byl říjen 2008, kdy se Android stal open-source projektem a jeho zdrojové kódy si tak mohl stáhnout kdokoli.

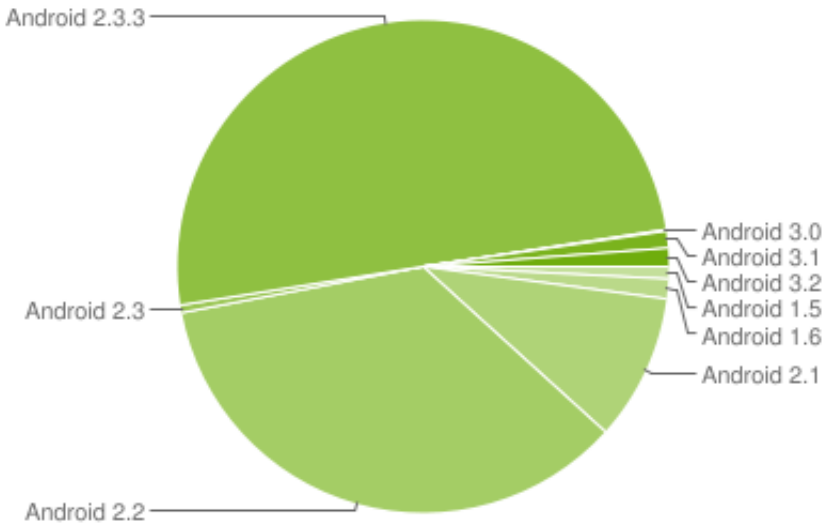
2.2 Verze Androidu

Jako každý operační systém, také Android prochází různými stádii vývoje. Na obrázku 2.1 lze vidět poměr jednotlivých verzí platformy Android platný k 1. prosinci 2011. Oproti minulým měsícům je zde razantní posun a přibývá telefonů s verzí 2.3 a novější. Při vývoji aplikací, které jsou součástí této práce, bude kladen důraz na použitelnost hlavně u verze 2.3 a novějších.

2.3 Architektura platformy

Architektura platformy Android se skládá z 5ti hlavních částí. Jsou to aplikace, aplikační rozhraní, knihovny, android runtime (běhové prostředí) a linuxové jádro.

[1] mobilní telefony s operačním systémem
[2] Software Development Kit
[3] aplikace, která umožňuje nákup a stahování dalších aplikací



Obrázek 2.1: Podíly jednotlivých verzí Androidu [11]

Aplikace – Android v základu obsahuje několik nejdůležitějších aplikací jako je webový prohlížeč, kalendář, spráce kontaktů aj. Aplikace jsou implementovány v jazyce Java.

Aplikační rozhraní – Umožňuje vytváření nových aplikací. Jedním z hlavních principů je možnost nahradit téměř jakoukoli aplikaci jinou, což dává uživateli velkou možnost upravit si systém přesně podle svých představ. Systém se tváří jako množina komponent, které mohou být nahrazovány.

Systémové knihovny – Jedná se o C/C++ knihovny, které jsou přeloženy do nativního[4] kódu. Tyto knihovny zprostředkovávají volání aplikačního rozhraní nad linuxovým jádrem. Mezi hlavní knihovny patří:

- libc – standardní knihovny jazyka C
- Multimedia Framework – knihovny pro práci s médii, jsou zde například kodeky pro přehrání videa
- WebKit – jádro webového prohlížeče
- SQLite – knihovny pro práci s jednoduchými databázemi
- SGL, OpenGL ES, Surface Manager – knihovny pro práci s grafikou
- FreeType – knihovna pro rvykreslování bitmapového a vektorového písma

Android runtime – Android obsahuje sadu hlavních knihoven (core libraries), které umožňují využít většinu z funkcionality jazyka Java. Každá aplikace má svůj vlastní proces s vlastní instancí ve virtuálním stroji, zvaném též Dalvik. Dalvik je vytvořen výhradně pro Android, protože jsou na tento systém kladeny specifické požadavky, jako je třeba nízká paměťová náročnost. Java aplikace jsou převedeny do formátu ".dex"(Dalvik Executable) a následně Dalvikem spouštěny. Díky tomuto přístupu je Android málo závislý na HW, na kterém běží, což mu dává určitou výhodu v přenositelnosti mezi různými výrobci, velikostmi displejů, HW konfiguracím apod.

[4] kód dané platformy

Linuxové jádro – Android je navržen na Linuxovém jádře verze 2.6. Obsahuje ovladače HW, správce paměti, procesů aj. Součástí je také určitá nadstavba [7], která zahrnuje meziprocesovou komunikaci (binder), práci se sdílenou pamětí (ashmem), správu napájení (wakelocks), "zabíjení" procesů při nedostatku paměti (low memory killer) atp.

Obrázek zobrazující schéma architektury je umístěn v příloze B.

Kapitola 3

Základy tvorby aplikací

Vyvíjet pro Android lze na systémech Windows, Linux i MAC OS X. Na všech těchto systémech lze provozovat Android SDK (viz následující podkapitola) a také vývojářský nástroj Eclipse[1]. Právě do Eclipse existuje rozšíření nazývané ADT[2], které značně usnadňuje vývoj.

3.1 Android SDK

SDK neboli Software Development Kit je soubor aplikací, utilit a knihoven pro vytváření, testování a další práci s Android aplikacemi. Mezi základní aplikace patří:

- aapt – slouží k vytváření *.apk[3] balíčků
- ddms – *Dalvik Debug Monitor Service*, jehož účelem je ladění a testování aplikace
- adb – *Android Debug Bridge*, který zaručuje spojení mezi PC a telefonem
- Monkey – Slouží k testování aplikací. Náhodně generuje operace (dotyk na displeji, příchozí SMS aj.) a testuje robustnost implementace.
- Draw9Patch – editor robustních obrázků typu *.9.png

Kromě SDK existuje také NDK[4], které umožňuje vývoj výkonově náročných částí aplikace přímo v nativním kódu. NDK je ovšem doplněk k SDK a nelze provozovat samostatně.

3.2 Eclipse ADT plugin

ADT plugin do Eclipse slouží pro snadnější vývoj Android aplikací. Mimo jiné nabízí správce virtuálních zařízení, možnost jednoduché aktualizace celého SDK či logcat, mechanismus pro sběr a zobrazování událostí v přístroji za účelem ladění aplikací [5].

Po vytvoření projektu jsou vygenerovány všechny položky, které projekt musí obsahovat. Toto značně usnadňuje vytváření projektů.

Správce virtuálních zařízení umožňuje vytvářet virtuální zařízení, tzv. AVD[5], s různými konfiguracemi a následně na nich testovat chování aplikací. Tato virtuální zařízení využívá

[1] http://www.eclipse.org/
[2] Android Development Tools – Android nástroje pro vývoj
[3] Android Application Package – balíček Android aplikace
[4] Native Development Kit – soubor aplikací pro vývoj v nativním kódu
[5] Android Virtual Device – Virtuální Android zařízení

emulátor, který dokáže vyvolat situace jako příchozí zpráva, příchozí hovor, změna GPS polohy a další.

3.3 Základní komponenty aplikace

Aplikace v Androidu se dá definovat jako soubor komponent. Hlavní 4 druhy [2] jsou:

- Activity – Aktivita
- Content Provider – Zdroj obsahu
- Service – Služba
- Broadcast Receiver – Všesměrový přijímač

3.3.1 Aktivita

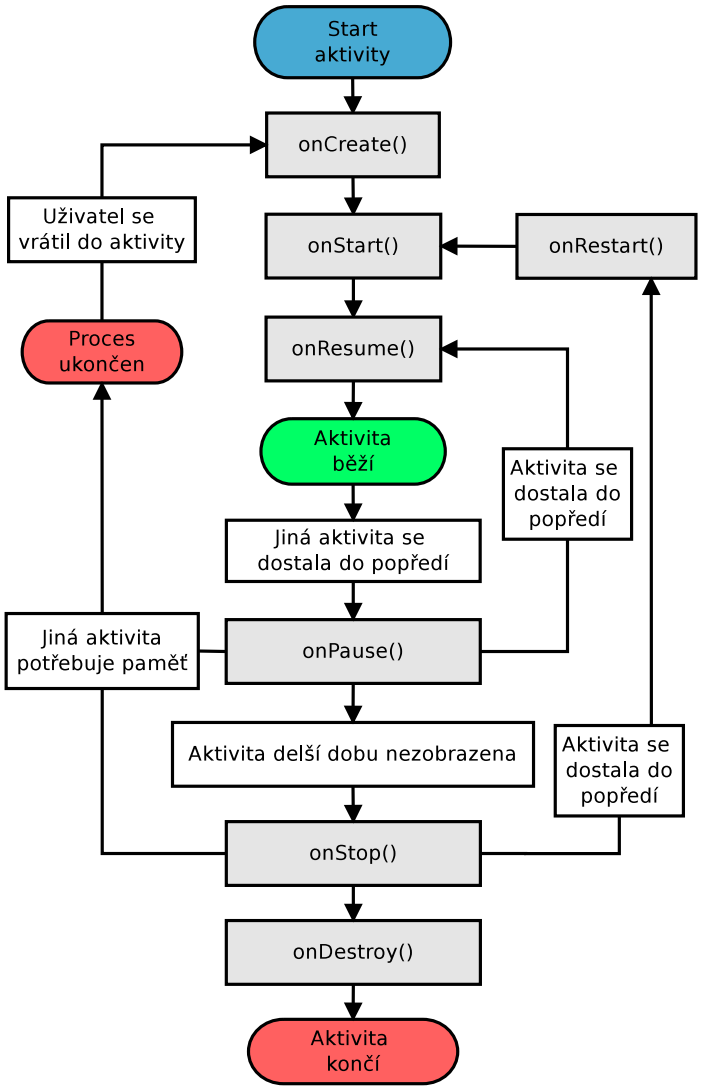
Aktivita slouží k zobrazení rozhraní, pomocí kterého může uživatel komunikovat s aplikací. Aktivita je složena z několika prvků uživatelského rozhraní nazývaných Views a zpravidla odpovídá jedné obrazovce aplikace. Aktivita při vzniku dostane přiřazenu položku v tzv. "back stacku", volně přeloženo "zpětný zásobník", který slouží k uchování posloupnosti aktivit v pořadí, jak mezi nimi uživatel přecházel. Pokud je volána další aktivita, stará je pozastavena a nová dostane přidělen vrchol zásobníku. V případě, že v této situaci uživatel stiskne tlačítko ZPĚT, vrchol zásobníku je uvolněn a systém se vrací k předešlé aktivitě. Aktivita se vytváří děděním třídy `android.app.Activity` a definováním alespoň metody *OnCreate()*.

V souvislosti s tímto, je nutno zmínit také `Tasks` neboli Úlohy. Úloha seskupuje aktivity vykonávané v rámci jedné aplikace resp. činnosti. Při startu aplikace je její úloha přenesena do popředí. Pokud aplikace instanci úlohy nemá, protože od startu zařízení nebyla aplikace spuštěna, je pro ni vytvořena.

Životní cyklus aktivity

Na obrázku 3.1 lze spatřit životní cyklus Aktivity v Android systému. Aplikace během svého života prochází různými stavy, přičemž využívá volání metod [1]:

- onCreate() – Volá se po prvním spuštění aplikace. Zde je vhodné provést inicializaci aktivity.
- onStart() – Volá se předtím, než je aktivita zobrazena uživateli.
- onResume() – Volá se po tom, co je aktivita zobrazena uživateli.
- onPause() – Jiná aplikace se dostala do popředí.
- onStop() – Aplikace je delší dobu nezobrazena.
- onDestroy() – Volá se, pokud je aplikace systémem zrušena např. kvůli nedostatku paměti.
- onRestart() – Volá se, pokud byla aplikace zastavena a dostala se znovu do popředí.



Obrázek 3.1: Cyklus života aktivity [1]

Intent

Intent slouží pro start nových aktivit, jako prostředník pro komunikaci nebo pro provádění zadaných akcí [16]. Lze je rozdělit do dvou kategorií a to na "intenty" *implicitní* a *explicitní*.

Explicitní intent slouží např. pro volání aktivit, kde přesně říkáme, jakou činnost chceme provést. Explicitní intent je definován třídou `aktivity`, kterou voláme a třídou `komponenty`, která jej volá:

```
Intent intent = new Intent(this, NovaActivity.class);
startActivity(intent);
```

V případě implicitních intentů říkáme co vyžadujeme, ale nezáleží jakým způsobem se k výsledku dostaneme. Implicitní intent lze definovat `akcí`, `URI dat`, se kterými se má pracovat či např. `kategorií`. Příkladem kategorie může být CATEGORY_BROWSABLE sdružující aplikace, které umí pracovat s odkazy, jako je např. e-mailová adresa. Následující

kód demonstruje spuštění akce, kdy žádáme informace o první osobě v kontaktech:

```
Intent intent = new Intent(
Intent.ACTION_VIEW, Uri.parse("content://contacts/people/1"));
startActivityForResult(intent);
```

Po vykonání akce je volána funkce `onActivityResult()`, ve které lze zpracovat výsledky nebo volat intent, který opět vybere nejlepšího kandidáta pro zpracování získaných dat.

Aby mohla aplikace reagovat na implicitní "intenty", existuje tzv. Intent Filter. S jeho pomocí můžeme dát systému vědět, že naše aplikace zvládá např. prohlížení webových stránek. Intent Filtry se definují v Android Manifestu (kapitola 3.5) s využitím tagu *<intent-filter>* a vnořených tagů *<action>*, *<category>* a *<data>*.

3.3.2 Poskytovatel Obsahu

Poskytovatel obsahu (angl. Content Provider) je rozhraní pro sdílení dat mezi aplikacemi. Aplikace je tedy schopna, zveřejnit nějakou část nebo celý svůj obsah, aby jej mohly jiné aplikace číst případně měnit. Poskytování obsahu se provádí dvěma způsoby, a to děděním třídy `android.content.ContentProvider` nebo připojením nabízeného obsahu k již existujícímu CP.

Poskytovat obsah bez možnosti jej číst by bylo samozřejmě k ničemu, a proto existuje třída zvaná `ContentResolver`. Jelikož je práce s CP v podstatě práce s SQLite databází, nabízí řadu metod, které jsou pro databáze typické. Stručný popis jednotlivých metod je uveden v kapitole 3.4. Pro identifikaci poskytovatelů se používají tzv. `URI`[6], což je řetězec odpovídající zápisu *content://<identifikátor-poskytovatele>/<cesta-ke-zdroji>*. Příkladem může být poskytovatel přijatých SMS - "`content://sms/inbox`". Existují také URI konstanty, které umožňují zjednodušit zápis - "`Calendar.CONTENT_URI`".

Pro přečtení dat pomocí CP je tedy zapotřebí jejich URI, názvů datových polí jejichž informaci žádáme (zpravidla sloupce v tabulce) a datové typy těchto polí. Pokud žádáme konkrétní záznam, je možné navíc připojení identifikátoru záznamu.

3.3.3 Služba

Služba (angl. Service) je aplikační komponenta uzpůsobena pro běh déle trvajících operací, např. stahování souboru z internetu. Služba běží na pozadí a neposkytuje žádné uživatelské rozhraní. Existuje i speciální varianta běžící na popředí.

Stejně jako aktivita má i služba svůj životní cyklus. Služby se dají rozdělit na dva druhy:

- started service (spouštěné služby) - Služba je vytvořena voláním `startService()` a běží dokud není ukončena voláním `stopService()` nebo se neukončí sama voláním `stopSelf()`. Po ukončení je ze systému odstraněna.
- binded service ("vázaná" služba) - Služba je vytvořena voláním `bindService()`. Komunikace mezi komponentou, která službu vytvořila a službou samotnou je prováděna pomocí třídy `IBinder`, která tvoří rozhraní pro meziprocesovou komunikaci. Oproti started service umožňuje tento typ navázání spojení se službou více komponentám (klientům) najednou. Spojení se ukončí metodou `unbindService()`. Odstranění služby je provedeno, pokud už není navázáno spojení s žádným klientem. Není proto nutné explicitně službu ukončit např. voláním `stopSelf()`.

[6] Uniform Resource Identifier

Tyto dva druhy ovšem nejsou jednoznačně výlučné. Jde spíše o dva principy jak mohou být služby užity. Služba, která byla vytvořena pomocí `startService()` ,může být také např. připojena k určité komponentě pomocí `bindService()`.

3.3.4 Všesměrový přijímač

Všesměrový přijímač (angl. Broadcast Receiver) slouží k reakci na některé události, které mohou v systému nastat. Jde například o příchozí SMS nebo telefonní hovor. BR je definován v Android Manifestu (3.5) nebo programově. Je platný pouze po dobu provádění metody `onReceive()`.

3.4 Datová úložiště

Android podporuje několik možností jak uchovávat perzistentní data. Výběr správného typu závisí na účelu dat. Principy ukládání dat:

- Shared Preferences – úložiště pro jednoduché záznamy klíč-hodnota
- Internal Storage – vnitřní paměť telefonu
- Extrenal Storage – externí úložiště, zpravidla paměťová karta
- SQLite Database – úložiště pro strukturovaná data přístupná pouze pro danou aplikaci
- Network Storage – Webové úložiště dat. Pro práci s tímto typem slouží balíčky *java.net.** a *android.net.**.

3.4.1 Vnitřní úložiště

Uložená data jsou standardně přístupná pouze aplikaci, které data přísluší. Toto chování lze potlačit při vytváření souboru. Příklad zápisu řetězce do souboru v interní paměti:

```
String SOUBOR = "soubor.txt";
String text = "nejaky text";

FileOutputStream fos = openFileOutput(SOUBOR, Context.MODE_PRIVATE);
fos.write(text.getBytes()); // zapis dat do souboru
fos.close();
```

Oproti zápisu do souboru je handler na soubor získán pomocí funkce `openFileInput()`. Po odinstalování aplikace jsou tato data smazána.

3.4.2 Externí úložiště

Data zde uložená jsou přístupná všem aplikacím i uživateli, jsou tedy sdílená. Před jakoukoli prací s externí pamětí je nutné otestovat, zdali je paměť dostupná. Uživatel mohl připojit svou SD kartu k PC jako velkokapacitní úložiště, což by zapříčinilo znepřístupnění karty v telefonu. Kontrolní mechanismus poskytuje funkce `getExternalStorageState()`, jejíž návratovou hodnotou je informace o stavu externího úložiště.

Ukládání dat na externí úložiště se liší také podle jejich účelu. Pokud jsou zde data uložena za účelem úspory vnitřní paměti a slouží pouze pro potřeby aplikace samotné, ačkoli k nim mají přístup i ostatní aplikace, zpravidla se nacházejí v adresáři

/Android/data/<název balíčku>/files. Stejně jako u interní paměti jsou tato data po odinstalování aplikace smazána.

Pro ukládání dat, která slouží nejen aplikaci samotné, ale jsou užitečná i pro ostatní aplikace nebo pro uživatele, je v kořenovém adresáři externí paměti předpřipraveno několik složek s názvy *Music/*, *Download/*, *Pictures/* aj. Tyto složky jsou vhodné k ukládání hudebních skladeb, stažených položek či obrázků.

3.4.3 SQLite databáze

Android disponuje plnou podporou SQLite databází. Doporučenou technikou k vytvoření databáze je vytvoření podtřídy třídy `SQLiteOpenHelper`. Pro práci s databází je dále nutné volat metodu `getWritableDatabase()` nebo `getReadableDatabase()` pro čtení resp. pro zápis do databáze. Návratovou hodnotou obou těchto funkcí je objekt typu `SQLiteDatabase`, který reprezentuje samotnou databázi. Mezi základní metody pro práci s databází patří:

- *query()* – přečtení dat z databáze, návratová hodnota je tzv. `Cursor`, který by se dal přirovnat k ukazateli na tabulku
- *insert()* – vložení dat do databáze
- *update()* – aktualizace dat v databázi
- *delete()* – smazání vybraných dat z databáze

V Android SDK se také nachází nástroj *sqlite3*, jehož účelem je prohlížení tabulek, vykonávání SQL příkazů a provádění dalších užitečných funkcí nad SQLite databázemi. Umožňuje tím simulovat různé situace v databázi.

3.5 Android Manifest

AndroidManifest je XML dokument, který se nachází v kořenovém adresáři každého balíčku. Manifest obsahuje základní informace o aplikaci. Mimo jiné se v něm nacházejí tyto informace:

- Pojmenování balíčku aplikace. Pokud je aplikace vkládána na Android Market, musí být tento název jedinečný vůči všem ostatním aplikacím.
- Popis všech použitých komponent - aktivity, služby, poskytovatelé obsahu (třída `Content Provider`) a všesměrové přijímače (třída `Broadcast Receiver`). Deklarace těchto komponent dává systému informaci, kdy a za jakých podmínek má být ona komponenta spuštěna.
- Definice oprávnění, které uživatel aplikaci udělí při instalaci. Pokud uživatel oprávnění neudělí, aplikace se nenainstaluje.
- Určuje minimální verzi SDK, které aplikace podporuje.

Jako příklad Android Manifestu je zde uveden kód 3.1. Popsaná aplikace se jménem umístěným v *@string/app_name*, podporuje již API verze 3 (Android 1.5), obsahuje jedinou aktivitu pojmenovanou *MyActivity* a má právo číst a psát do kontaktů.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="bp.sztefekl.myactivity"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="3" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".MyActivity" >
            <intent-filter >
                <action
                  android:name="android.intent.action.MAIN"/>
                <category
                  android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
</manifest>
```

Kód 3.1: Příklad jednoduchého manifestu

3.6 Zdroje

Zdroje (angl. Resources) je v Androidu souhrnné pojmenování pro statická data, která jsou přiložena k aplikaci. Nachází se v adresáři *res/* v kořenovém adresáři projektu. Hierarchie vnořených adresářů je uvedena v tabulce 3.1. Pro takto specifikovaná data je vytvořena speciální třída *R*, která obsahuje konstantní identifikátory jednotlivých zdrojů. Tato třída je generována automaticky, a proto vždy aktuální. Instanci zdroje lze následně v aktivitě získat pomocí jeho identifikátoru `R.typ.jméno` a funkce `getResource()`.

Telefony s AndroidOS disponují různými rozlišeními, velikostmi displejů, jazykovými sadami, HW schopnostmi apod. Proto existují alternativní zdroje. Každá aplikace se při instalaci snaží co nejvíce přizpůsobit telefonu, na který je instalována. Příkladem může být lokalizace aplikace do češtiny. V základu se ve složce *res/values/* zpravidla nachází soubor *strings.xml* se všemi řetězcovými konstantami v anglickém jazyce. Pokud programátor vytvoří složku *res/values-cs/* se souborem *strings.xml*, kde jsou všechny řetězce přeloženy do češtiny, bude při instalaci aplikace do telefonu s českým prostředím upřednostněna česká lokalizace před anglickou. Pokud budou některé řetězce v české lokalizaci chybět, jsou nahrazeny anglickými. Nedojde tedy k situaci, že by některý řetězec nebyl zobrazen.

Tento způsob personalizace lze také využít pro optimalizaci. Programátor může vytvořit obrázky s různým rozlišení pro různé velikosti displejů. Tím zabrání tomu, aby systém zbytečně nahrával do operační paměti obrázky s velkou velikostí, když je displej nedokáže zobrazit v plné kvalitě.

Adresář	**Popis obsahu**
anim/	XML soubory pro popis animací.
color/	XML soubory definující skupiny barev pro komponenty GUI. Umožňují např. vytvořit skupinu popisující barvu tlačítka pro jeho různé stavy (stisknuto, zaměřeno aj.).
drawable/	Bitmapové (.png, .9.png, .jpg, .gif) a XML soubory obsahující resp. popisující obrázky, ikony aj.
layout/	XML soubory definující rozvržení uživatelského rozhraní.
menu/	XML soubory specifikující nabídky, kontextová menu a podmenu aplikace. Vytvořit menu lze několika způsoby, nemusí se zde nacházet popisy všech menu.
raw/	Vlastní soubory libovolného typu.
values/	XML soubory definující jednoduché konstanty jako jsou řetězec, barva či číslo. Protože je každý zdroj definován svým vlastním XML elementem, je možné vložit všechny položky do jediného souboru. Není to ale nejlepší cesta, protože se zdroje stávají nepřehlednými. Pro soubor s textovými řetězci se nejčastěji používá pojmenování *strings.xml*, pro soubor s textovými poli *arrays.xml*, pro soubor se specifikací barev *color.xml* (narozdíl od adresáře *color/* se zde specifikují pouze jednotlivé barvy jako konstanty), *dimens.xml* pro soubor s rozměry (Dimensions) a konečně *styles.xml* pro soubor se styly.
xml/	Dodatečné XML soubory. Narozdíl od souborů v adresáři *raw/* je lze snadno číst, protože je jejich obsah strukturovaný, což o surových datech platit nemusí.

Tabulka 3.1: Popis adresářů zdrojů

Kapitola 4

Grafické uživatelské rozhraní

GUI Androidu je uzpůsobeno k dotykovému ovládání telefonu, tabletu či jiného zařízení. Narozdíl od klasické Javy, která podporuje rozhraní AWT[1] či Swing, je zde vlastní rozhraní.

Uživatelské rozhraní aplikace je složeno pomocí *View* (pohled) a *GroupView* (skupina pohledů) objektů. Podporováno je mnoho druhů těchto pohledů a skupin, všechny jsou ale potomky třídy `android.view.View`. Uživatelské rozhraní spojováním pohledů a skupin pohledů vytváří stromovou hierarchii. Předání stromové struktury pohledů pro vykreslení na displeji lze vykonat voláním metody `setContentView()`.

Důležitou roli při vytváření rozhraní hraje vhodné použití jednotek. Jednotky, které lze v Androidu použít zobrazuje tabulka 4.1.

Zkratka	**Název**	**Popis**
px	pixel	Jeden bod displeje.
mm	milimetr	Velikost jednoho milimetru na displeji.
in	inch	Velikost jednoho palce na displeji.
pt	point	Velikost 1/72 palce (inch) na displeji.
dp (dip)	Density-Independent Pixel	Relativní jednotka založená na hustotě pixelů na displeji.
sp	Scale-Independent Pixel	Podobné jako dp, ale používá se pro určení velikosti písma, ale navíc zohledňuje uživatelské nastavení.

Tabulka 4.1: Rozměrové jednotky užívané v Androidu [15]

Velikost 1dip odpovídá 1px na 160 dpi[2] displeji. Se vzrůstajícím dpi displeje se také zvyšuje hodnota dip. Tuto závislost znázorňuje následující rovnice:

$$\text{Počet } px = dip * (dpi/160) \tag{4.1}$$

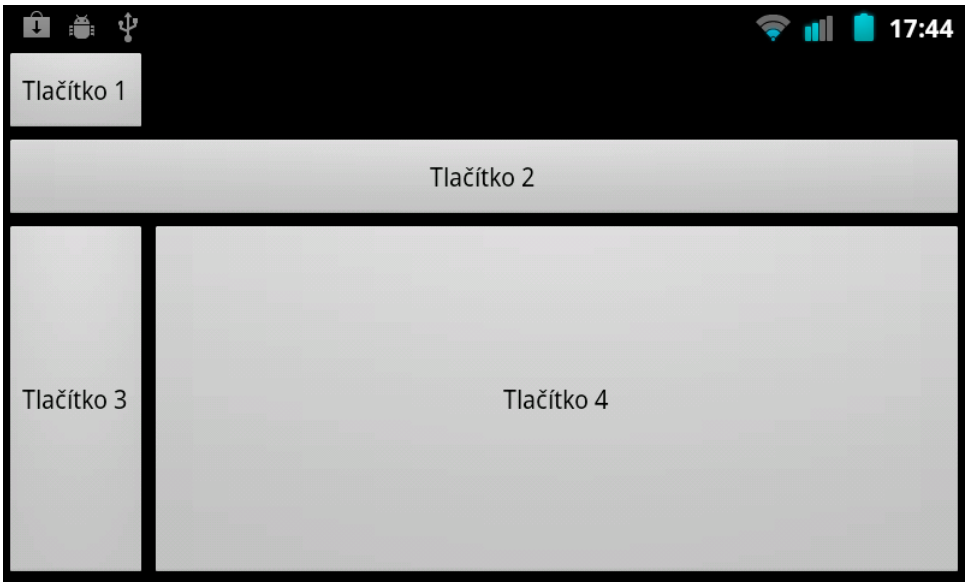
4.1 Layout

Layout určuje způsob rozvržení komponent uživatelského rozhraní na displeji zařízení. Lze je vytvářet pomocí XML zápisu nebo přímo v kódu programově. Samozřejmě

[1] Abstract Window Toolkit – část Javy vhodná k vytváření grafického uživatelského rozhraní
[2] Dots per Inch - počet pixelů na velikost 1 palce

lze oba přístupy kombinovat, tedy například navrhnout základní rozvržení pomocí XML a přidat další prvky programově. Pro určení základní výšky a šířky prostoru, který layout, resp. kterýkoli jiný grafický prvek zabírá, slouží atributy `android:layout_height` a `android:layout_width`. Tyto atributy mohou nabývat hodnot `wrap_content`, což zmenší prvek na nejmenší možnou velikost a `match_parent`, což zvětší plochu displeje, kterou prvek zabírá, na největší možnou. Android ve verzi 2.1 a nižší používal pro `match_parent` pojmenování `fill_parent`. Různé kombinace těchto atributů zobrazuje obrázek 4.1. Jsou na něm vyobrazena 4 tlačítka, kde:

- Tlačítko 1 disponuje hodnotou `wrap_content` jak pro nastavení šířky, tak výšky,
- Tlačítko 2 má šířku nastavenu na `match_parent` a výšku na `wrap_content`,
- Tlačítko 3 má opačné atributy v porovnání s tlačítkem 2,
- Tlačítko 4 má jak výšku, tak šířku nastavenu na `match_parent`, zabírá tedy největší možnou plochu.



Obrázek 4.1: Nastavení výšky a šířky grafických prvků

Existuje 5 základních typů layoutů:

- LinearLayout – Toto rozvržení organizuje komponenty do řádků. Pomocí atributu `android:orientation` lze určit, má-li se každá další komponenta umisťovat na nový řádek (atribut je roven hodnotě `horizontal`) nebo je požadavkem vložit komponenty na jeden řádek (hodnota `vertical`).
- FrameLayout – Je uzpůsoben pro zobrazení jediného prvku. Při vložení několika prvků se budou překrývat.
- TableLayout – Rozvržení chovající se jako tabulka. Každý řádek tabulky může obsahovat 0 a více buněk (cells), jediná buňka obsahuje jeden *View* objekt. Takovým objektem ale může být i další layout, tedy dalších několik komponent.
- RelativeLayout – Definuje rozložení jednotlivých komponent vzhledem ke komponentám ostatním. Například určením, že tlačítko A se má nacházet vpravo od tlačítka B a zároveň pod textem T. Systém následně rozvrhne rozložení tak, aby byly všechny podmínky splněny, samozřejmě za předpokladu, že taková kombinace existuje.

- AbsoluteLayout – Absolutní rozvržení pozic komponent na displeji pomocí XY souřadnic. Použití tohoto rozvržení se doporučuje pouze ve specifických případech, jinak může docházet k chybám s překrýváním aj.

4.2 Základní komponenty

Android poskytuje mnoho komponent grafického uživatelského rozhraní, které si navíc může programátor mnoha způsoby upravovat pro své potřeby. Tato kapitola zmiňuje ty nejdůležitější a nejpoužívanější.

4.2.1 Tlačítko

Tlačítko (angl. Button) je komponenta reagující na klik. Ten může být proveden buď dotykem dotykové plochy displeje nebo potvrzovacím tlačítkem telefonu, pokud jím disponuje. Reakce na kliknutí se provádí zpravidla dvěmi způsoby:

- Určením metody, která má být zavolána při kliknutí na tlačítko, přímo v XML souboru, kde je tlačítko definováno. Do atributu `android:onClick` přiřadíme jméno metody.
- Vytvořením tzn. "listeneru", jakožto posluchače na událost, v tomto případě tedy stisk tlačítka.

Kromě posluchače, který reaguje na prostý klik na tlačítko, existují ještě další. Příkladem může být `View.OnLongClickListener`, tedy reakce na dlouhé stisknutí, kterou demonstruje kód 4.1. V případě, kdy programátor specifikuje pro jedno tlačítko posluchače na jednoduché kliknutí i dlouhé kliknutí, musí být návratovou hodnotou metody *onLongClick()* hodnota *true*. Jedná se o potvrzení, že událost byla zpracována a není ji potřeba přeposílat na další zpracování. V případě navrácení hodnoty *false*, by zároveň došlo k vyhodnocení akce také jako klasického kliknutí, což je nežádoucí.

```
btn.setOnLongClickListener( new View.OnLongClickListener() {

  @Override
  public boolean onLongClick(View v) {
    // akce vykonana po dlouhem stisku tlacitka
    return true;
  }
});
```

Kód 4.1: Přiřazení posluchače reagujícího na dlouhý stisk tlačítka

Třída `Button` má několik potomků. Mezi ně patří `Checkbox`, tedy zaškrtávací tlačítko či `RadioButton`, které je podobné předešlému, ale lze jej seskupovat do tzv. `RadioGroup` a dát uživateli možnost zvolit některé tlačítko ze skupiny. Dalším zástupcem je `ToggleButton`, které má také podobné chování jako `CheckBox`, ale po změně stavu se mění jeho text.

4.2.2 EditText a TextView

Jak již název těchto komponent napovídá, slouží k manipulaci s textem. `EditText` jako políčko pro načtení textového vstupu a `TextView` k zobrazení textového popisku, v jiných prostředích často nazýván *label*.

Každá z těchto komponent má svůj stěžejní atribut. U `TextView` je to atribut `android:text`, což je určení zobrazovaného textu. Tento text lze později programově měnit pomocí metody *setText()* instance popisku. Samozřejmostí je také změna fontu, barvy pozadí, barvy textu nebo velikosti textu. Oproti tomu `EditText`, klade hlavní důraz na jednoduché ověřování vkládaného textu pomocí atributu `android:inputType`. Lze zadat hodnoty jako *textPassword* (vložený text skryt za hvězdičky), *phone* (povoleny pouze čísla a některé speciální znaky jako # nebo *) nebo například *date*, je-li žádáno datum.

4.2.3 Notifikace

Notifikace slouží k jednorázovému informování uživatele formou jednoduché zprávy. Takto příchozí zprávy se nacházejí v oblasti, kterou je možné zobrazit tažením horní lišty systému směrem dolů. Zpravidla zde bývají upozornění na příchozí zprávy, nové emaily, dokončené instalace aj.

Notifikace lze vytvářet pomocí třídy `Notification`. U notifikace lze specifikovat také dodatečné projevy, kupříkladu zvuková signalizace nebo vibrace. U některých notifikací se hodí definovat akci, která se vykoná při kliknutí na jejich tělo. Toto chování lze specifikovat pomocí třídy `PendingIntent` (odložený intent). Ten zabalí klasický intent a započne jeho vykonávání jako reakci na nějakou akci, v tomto případě kliknutí na notifikaci. Základní forma notifikace zobrazuje titulek a obsah. Je ale možné navrhnout vlastní schéma toho, jak má notifikace vypadat. Lze také nastavit několik druhů příznaků, jako např. `Notification.FLAG_AUTO_CANCEL`, což zapříčiní odstranění notifikace, pokud na ni uživatel kliknul.

Pro správu notifikací slouží třída `NotificationManager`. Reference na tento objekt lze získat voláním metody *getSystemService()* s parametrem `Context.NOTIFICATION_SERVICE`.

4.2.4 Dialog a Toast

Obě tyto třídy slouží k zobrazení upozornění, které překryje nějakou část okna s aktuální aktivitou. `Toast` umožňuje několikasekundové zobrazení kratší zprávy v dolní části displeje. Používá se hlavně při zobrazení nepříliš důležitých zpráv, pokud zprávu uživatel nezaznamená, nejedná se o velký problém. Opačným případem je `Dialog`, kdy dochází k zobrazení nového okna, které žádá uživatelovu reakci nebo přímo informuje o nějaké prováděné akci. Dialogy lze rozdělit do několika kategorií [3]:

- AlertDialog – Dialog, kterému lze přiřadit až 3 tlačítka a umožňuje od uživatele vyžadovat interakci, například potvrzení nějakého kroku. Stejně tak může být obsahem dialogu seznam, který umožňuje uživateli výběr jedné z nich.
- ProgressDialog – Tento dialog poskytuje uživateli informaci o právě probíhající činnosti. Dialog zobrazuje stav provádění pomocí plnícího se pruhu, který zobrazuje aktuální pokrok při provádění činnosti, nebo pomocí animace s rotujícím kolečkem.
- DatePickerDialog – Dialog pro výběr data. Na dialogu jsou zobrazeny posuvníky pro intuitivní a snadný výběr.
- TimePickerDialog – Totožný princip jako u předešlého dialogu, pouze s rozdílem, že tento dialog podporuje výběr času.

4.2.5 Menu a nabídky

Menu jsou jednou ze základních komponent většiny uživatelských rozhraní. Nabídky lze rozdělit do tří kategorií:

- Options menu (hlavní nabídka) – Toto menu se zobrazí ve spodní části displeje při kliknutí na fyzické tlačítko *Menu* přístroje. Toto platí pouze pro zařízení disponující Androidem ve verzi 2.3.3 a nižší. Novější verze nadále toto tlačítko nevyžadují. Místo toho přímo na displeji zobrazují komponentu zvanou `ActionBar`, kde mohou být položky menu zobrazeny, pokud byly nastaveny jako "action items". V opačném případě se jejich výčet skrývá pod symbolem tří vertikálně umístěných teček tzv. "overflow button".

  Menu ve verzi 2.3.3 a nižších dokáže najednou zobrazit až 6 položek. Pokud programátor vytvořil položek více, šestá pozice v menu se začne chovat jako rozbalovací podmenu zobrazující výčet ostatních položek.
- Context menu (kontextová nabídka) – Jedná se o plovoucí menu, které se zobrazí při dlouhém stisku prvku grafického uživatelského rozhraní, se kterým je propojeno. Android ve verzi 3.0 a vyšší opět podporuje zobrazení tohoto menu v `ActionBaru`. Kontextové menu lze přiřadit všem potomkům třídy `View`.
- Popup menu (vyskakovací nabídka) – Vyskakovací nabídka je velmi podobná kontextové nabídce, ale její obsah se netýká žádné konkrétní položky. Tato nabídka je dostupná pouze v Androidu verze 3.0 a vyšší.

Nabídky lze vytvářet pomocí XML nebo programově. Při požadavku vytvoření menu, zpravidla po stisknutí příslušného tlačítka, dochází k volání metody *onCreateOptionsMenu()* resp. *onCreateContextMenu()*, kde je prostor pro vytvoření samotné nabídky. V případě vytváření menu pomocí XML souboru se používá metoda *inflate()* třídy `MenuInflater`. Oproti tomu programově pomocí metody *add()* instance menu, která je předána do výše zmíněných funkcí jako parametr. Oba tyto přístupy demonstruje kód 4.2.

Reakce na stisknuté klávesy lze uskutečnit v metodě *onOptionsItemSelected()* resp. *onContextItemSelected()*.

```
// vytvareni menu programove
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add("Polozka 1");
    menu.add("Polozka 2");
    return true;
}

//vytvareni menu pomoci XML
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Kód 4.2: Vytváření menu programově a pomocí XML

4.2.6 ListView

`ListView` lze částečně zařadit mezi layouty, protože často bývá použit jako jediný prvek v aktivitě. Tato komponenta vytváří seznam položek, který mu byl zprostředkován pomocí tzv. adaptéru. Adaptér si vnitřně ukládá seznam položek a pomocí metody *setListAdapter()* dojde k připojení k `ListView` komponentě.

Existuje několik druhů adaptérů, v závislosti na zdroji, který jim poskytnul seznam položek, a také na způsobu mapování položek do seznamu. V závislosti na typu zdroje můžeme určit dva základní druhy adaptérů. Prvním typem je `ArrayAdapter`, kterému jako zdroj dat vyhovuje klasické pole s položkami typu `String`. Druhým je `CursorAdapter`, jehož zdrojem bývá `Cursor`, tedy seznam položek získaných z databáze. Tyto adaptéry se vyznačují mapováním 1 : 1, jedna položka zdrojového seznamu na jeden řádek `ListView`.

Jednotlivé řádky `ListView` lze specifikovat pomocí vlastního XML souboru. Není proto problém vytvořit řádek s několika textovými položkami. Z tohoto důvodu existují pokročilejší adaptéry, které mapují X zdrojových položek na X cílových položek. Takové adaptéry mají předponu `Simple`. Budiž nám příkladem `SimpleCursorAdapter`. V jeho konstruktoru se nachází parametry `String [] from` a `Int [] to`. V poli "`from`", specifikujeme sloupce tabulky a v poli "`to`" grafické komponenty cílového řádku. Kód 4.3 demonstruje tyto konstrukce.

```
// vyber sloupcu z tabulky
String[] columns = new String[] { People.NAME, People.NUMBER };

// vyber komponent radku, do kterych se bude mapovat
int[] to = new int[] { R.id.text_jmeno, R.id.text_cislo };

SimpleCursorAdapter mAdapter = new SimpleCursorAdapter(
            this, R.layout.listview_radek, cursor, columns, to);
```

Kód 4.3: Mapování dat z tabulky do adaptéru

Adaptéry lze použít ve více oblastech než pouze při manipulaci s `ListView`. Jako příklady lze uvést např. `Spinner`, z jiných prostředí známý též jako combobox, nebo `Gallery`, galerie obrázků. Nezřídka bývá adaptér označován jako "man in the middle", tedy prostředník, který propojuje prvky grafického rozhraní s datovými zdroji.

Kapitola 5

Práce se senzory a widgety

Tato kapitola popisuje možnosti Android OS v oblasti senzorů a možnosti jednotlivých senzorů. Dále je uveden účel widgetů a jejich vytváření.

5.1 Senzory

V rámci tohoto projektu budu popisovat senzory zařízení HTC HD2[1], který sice v původním stavu Android neobsahuje, ale díky komunitě se tento systém na telefon dostal. Toto zařízení disponuje následujícími senzory:

- Akcelerometr
- Senzor magnetického pole
- Polohový senzor
- Proximity senzor
- Světelný senzor

Pro práci se senzory slouží tyto třídy:

- SensorManager - Slouží k vytváření instance služeb pracujících se senzory. Poskytuje metody pro zjištění dostupných senzorů, pro registraci a odregistraci posluchačů (*SensorEventListener*). Úsek kódu 5.1 popisuje způsob získání seznamu dostupných senzorů.
- Sensor - Slouží k vytváření instance specifického senzoru. Poskytuje metody pro zjišťování parametrů senzoru.
- SensorEvent - Vytváří události spojené s prací senzoru. Událost obsahuje tyto informace: samotná data ze senzoru, typ senzoru, který událost vyvolal, ukazatel přesnosti dat a časovou známku dat.
- SensorEventListener - Slouží k vytvoření 2 callback funkcí[2], které jsou volány v případě změny přesnosti nebo výstupních dat senzoru.

[1] http://www.htc.com/cz/help/htc-hd2/

[2] funkce, která je volána jako zpětná vazba na určitou akci

```
private SensorManager mSensorMan;
...
mSensorMan = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> deviceSensors = mSensorMan.getSensorList(Sensor.TYPE_ALL);
```

Kód 5.1: Příklad získání seznamu senzorů

5.1.1 SensorEventListener

Třída SensorEventListener slouží k příjmu notifikací při změně výstupních dat senzoru. Pokud chce programátor průběžná data ze senzoru dostávat, musí si nejprve registrovat jejich odběr. Registrace probíhá přes instanci třídy `SensorManager` pomocí metody *registerListener()*. Parametry metody jsou reference na instanci třídy SensorEventListener, reference na instanci třídy Sensor a určení časového intervalu mezi přijímanými daty. Interval lze specifikovat pomocí předpřipravených konstant `SENSOR_DELAY_NORMAL`, `SENSOR_DELAY_UI`, `SENSOR_DELAY_GAME`, `SENSOR_DELAY_FASTEST` nebo vlastní hodnotou v mikrosekundách. Velikost intervalu není garantovaná a může se mírně lišit.

V implementaci posluchače musí být překryty dvě metody, *onSensorChanged* a *onAccuracyChanged()*. První zmíněná slouží jako callback funkce při přijetí nových dat ze senzoru, druhá informuje, že došlo ke změně přesnosti senzoru.

Když nejsou data senzoru dále potřebná, musí se jejich příjem odregistrovat metodou *unregisterListener()*. To bývá zpravidla v metodě *onPause()* hlavní aktivity když dochází k přenesení aplikace do pozadí.

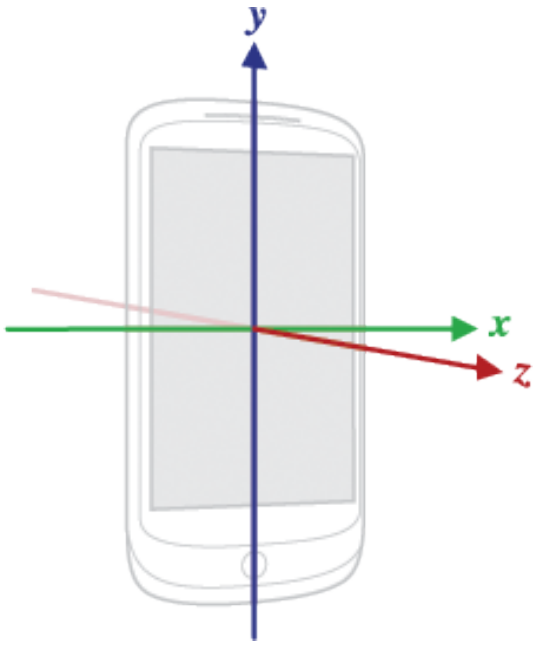
5.1.2 Akcelerometr

Akcelerometr zaznamenává zrychlení přístroje ve 3 různých osách. Orientace os vzhledem k zařízení jsou uvedeny na obrázku 5.1. Tento souřadnicový systém využívají také ostatní senzory. Součástí výsledných dat akcelerometru je i gravitační zrychlení Země. Reálné zrychlení zařízení je tedy potřeba přepočítat, např. s využitím polohového senzoru či senzoru magnetického pole. Pokud zařízení disponuje Androidem ve verzi 2.3 a vyšším, je softwarově dostupný tzv. lineární akcelerometr. U tohoto typu senzoru není přepočet potřeba a výsledné hodnoty jsou bez gravitačního zrychlení.

V základu je akcelerometr velice dobrý senzor pro zjišťování pohybů zařízení. Oproti ostatním polohovým senzorům využívá až 10 krát méně energie [8]. Výsledná hodnota senzoru je v jednotkách m/s^2.

5.1.3 Polohový senzor

Polohový senzor určuje míru naklonění/natočení telefonu vzhledem ke 3 osám. V souvislosti s možností změny orientace prostředí telefonu, *portrait* – na výšku a *landscape* – na šířku, je zde ale jedna potíž. Při změně orientace displeje se mění polohy os vzhledem k telefonu, ovšem polohy os senzoru, jako hardwarového zařízení, zůstávají nezměněny. Tento problém lze vyřešit buďto znemožněním uživateli změnit orientaci nebo využitím metody `SensorManager.remapCoordinateSystem()`, která provádí přepočet souřadnic. Právě kvůli tomuto problému se v současné době již příliš tento senzor nepoužívá a jeho použití k získání orientace bylo označeno jako zastaralé. K určení orientace nyní slouží kombinace výstupu z akcelerometru a senzoru magnetického pole Země.



Obrázek 5.1: Souřadnicový systém používaný senzorovým rozhraním [12]

5.1.4 Proximity senzor

Zvaný též senzor přiblížení. Používá se hlavně k indikaci toho, jestli je telefon při hovoru přiložen k uchu, což zapříčiní vypnutí displeje. Proximity senzor většiny přístrojů dokáže určit průměrnou vzdálenost od nejbližšího předmětu v centimetrech. Ostatní pouze určí, je-li předmět blízko/daleko.

5.1.5 Světelný senzor

Senzor monitorující úroveň okolního světla. Data tohoto senzoru není nutno nijak převádět či filtrovat jako u akcelerometru.

5.1.6 Senzor geomagnetického pole

Tento senzor monitoruje změny v magnetickém poli Země. Výstupní data jsou v jednotkách μT (mikroTesla) a určují magnetickou indukčnost pro všechny 3 osy souřadnicového systému, který je shodný se souřadnicovým systémem akcelerometru.

5.2 Widgety

Widget je souhrnné označení pro miniaplikace, které mohou být umístěny na jednu z ploch telefonu. Tyto miniaplikace poskytují obsah přímo, bez nutnosti spouštět přidruženou aktivitu (pokud existuje). Na jejich pozadí běží služba, která se stará o aktualizace a kontrolu stavu. Příkladem widgetu může být jednoduché ovládání MP3 přehrávače, indikátor nových emailů, zobrazení data a času atp. Uživatel je schopen widgety přidat buďto podržením prstu na zvolené ploše a vybráním možnosti Widgety (telefony s Androidem ve verzi 4.0 a nižší) nebo výběrem přímo z hlavního menu (telefony s Androidem ve verzi

4.0 a vyšší). Počet instancí jednoho typu widgetu není omezen, každý získá jednoznačný identifikátor.

5.2.1 Balíček s widgetem

Balíček widgetu musí obsahovat tyto základní komponenty:

- XML popis widgetu, také nazývaný jako *metadata widgetu*. Tato metadata určují jak často bude widget aktualizován, kolik plochy bude zaujímat a obsahují referenci na počáteční vzhled widgetu. Kód 5.2 znázorňuje příklad XML souboru definující metadata widgetu.
- Vzhled widgetu definovaný pomocí XML. Tento vzhled je možno později měnit, ať už pomocí jiného XML nebo programově.
- Java třídu, která je potomkem třídy `AppWidgetProvider`. Metoda *onUpdate()* této třídy je volána pokaždé po uplynutí zvoleného časového úseku.
- Android Manifest implementující `BroadcastReceiver`, jehož účelem je notifikace výše uvedené Java třídy, že došlo k vypršení časovače a widget má být aktualizován.

Kromě výše uvedených komponent, může balíček obsahovat další aktivity například pro konfiguraci. Reference na konfigurační aktivitu je poté součástí metadat widgetu.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dip"
    android:minHeight="72dip"
    android:initialLayout="@layout/widget_message"
    android:updatePeriodMillis="86400000"
 />
```

Kód 5.2: Příklad XML souboru s metadaty widgetu

5.2.2 Určení velikosti

Dalším důležitým faktorem při vytváření widgetu je určení jeho velikosti. Protože Android podporuje početné množství rozlišení displeje, pro určení velikosti se používá relativní jednotka `dip` (více tabulka 4.1). Na displeji s větším rozlišením budou mít widgety relativně stejnou velikost jako na displejích s menším rozlišením, budou ale zaujímat větší počet pixelů. Vlastní plochy telefonu jsou rozděleny do buněk, které tvoří maticovou strukturu. Vzorec 5.1 udává velikost widgetu v závislosti na počtu zabraných buněk [13].

$$\text{Velikost v } dip = (\text{Počet buněk} * 74dip) - 2dip \tag{5.1}$$

5.2.3 Životní cyklus widgetu

Protože je widget v podstatě všesměrový přijímač, vlastní metody jsou volány na základě doručování zpráv. Po vložení widgetu na plochu je spuštěna metoda `onEnabled()`, která přijímání zpráv aktivuje. V druhém kroku následuje volání metody `onUpdate()` a spuštění konfigurační aktivity, pokud je definována. Pokud uživatel widget z plochy odejmul přetažením na ikonu koše, metoda `onDeleted()` zaručí jeho uvolnění z paměti. Pokud byla smazaná instance zároveň poslední instancí daného typu widgetu, následují poslední fáze života a to deaktivace přijímání zpráv pomocí metody `onDisable()`.

Kapitola 6

Implementované aplikace

V této kapitole proberu postup implementace několika mých aplikací. První z nich je aplikace Orientace a Zrychlení, druhou 3D Kompas a poslední aplikace Tracker. V první části uvádím příklady použití senzorů a získávání orientace. Ve spojitosti s kompasem zde popisuji základy práce s prostředím Open GL ES a úpravu dat získaných senzory. V případě aplikace Tracker se věnuji práci s databází, se službami aj.

6.1 Orientace a Zrychlení

Tato aplikace poskytuje údaje o aktuálním zrychlení a náklonu v osách x/y/z. Využil jsem zde dat ze senzoru magnetického pole, akcelerometru a lineárního akcelerometru. Velkou část této aplikace jsem využil také v aplikaci následující, 3D Kompasu. Zdrojové kódy aplikace jsou umístěny na CD v adresáři */src/Orientace_Zrychleni*.

6.1.1 Základní parametry aplikace

Aplikace obsahuje pouze jedinou třídu, ve které je implementována veškerá logika. Na každý ze senzorů je registrován posluchač, přes který jsou aplikaci dodávana data. Senzor magnetického pole společně s akcelerometrem produkují orientaci přístroje, lineární akcelerometr poskytuje aktuální zrychlení. Orientace jednotlivých os vzhledem k přístroji zobrazuje obrázek 5.1.

6.1.2 Registrace a odregistrace posluchačů

Kód 6.1 představuje způsob registrace posluchače na senzor. Takto registrovaný posluchač je platný i v případě, že je aktivita přesunuta do pozadí. Musí se tedy odregistrovat v metodě *onPause()*. Pokud se aktivita opět vrátí do popředí, lze posluchače znovu registrovat v metodě *onResume()*.

6.1.3 Zjišťování orientace

Zjišťování orientace probíhá ve dvou fazích. První fáze spočívá ve zjištění senzorických dat a druhá v určení orientace podle získaných dat.

V případě, že již jsou data akcelerometru a senzoru magnetického pole dostupná, jsou předána statické metodě *getRotationMatrix()* třídy `SensorManager`. Jedním z argumentů metody je také pole devíti či šestnácti položek typu `float`, tzv. rotační matice. Tato matice

je při volání metody naplněna a lze ji předat další statické metodě *getOrientation()*, která produkuje požadované hodnoty náklonu pro každou z os.

```
SensorManager sensorMan = (SensorManager) getSystemService(SENSOR_SERVICE);
Sensor accelerometer = sensorMan.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

SensorEventListener accelerometerListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
         //...
        }
    };
sensorMan.registerListener(accelerometerListener,
                accelerometer, SensorManager.SENSOR_DELAY_UI);
```

Kód 6.1: Registrace posluchače senzoru

6.2 3D Kompas

Aplikace 3D Kompas umožňuje pomocí akcelerometru a senzoru magnetického pole určovat orientaci zařízení vzhledem k severnímu pólu země. Model střelky je vytvořen a vykreslován pomocí rozhraní *OpenGL ES*. Červená část střelky ukazuje magnetický sever. Základní aktivitu zobrazuje obrázek 6.1. Zdrojové kódy aplikace jsou umístěny na přiloženém CD v adresáři *src/3DKompas*.

6.2.1 Základní komponenty

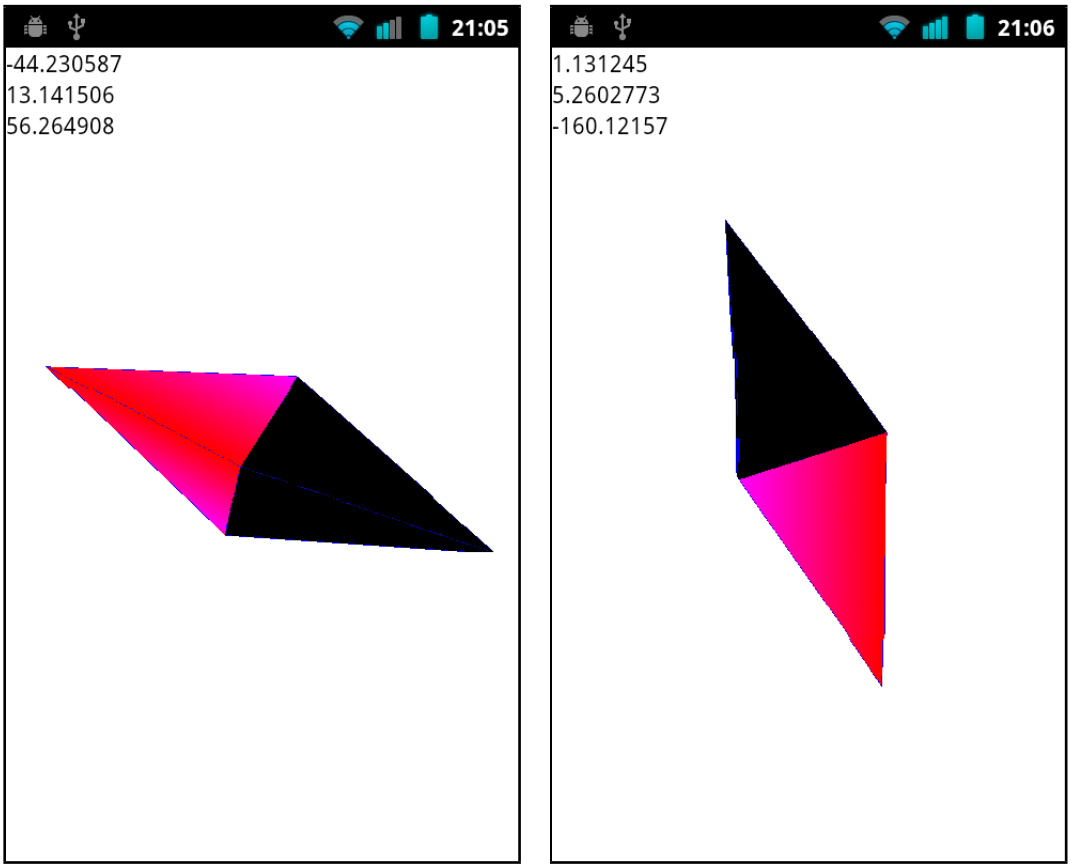
Aplikace se skládá z 3 základních tříd. První třída `CompassActivity` je potomkem třídy `Activity` a jejím účelem je vytvoření, práce se senzory, zpracování dat přijatých senzory a předávání vypočtených hodnot vláknu, které zajišťuje vykreslování. Druhá třída `Renderer`, zděděná ze třídy `android.opengl.GLSurfaceView.Renderer`, slouží k inicializaci OpenGL ES rozhraní a vykreslování modelu specifikovaného třetí třídou `CompassModel`.

6.2.2 OpenGL ES

Jak již bylo předesláno v kapitole 2.3, Android disponuje rozhraním OpenGL ES pro práci s 2D a 3D grafikou. Jde o klasické OpenGL, ochuzené o některé funkce jako např. *glBegin()* včetně jeho párové funkce *glEnd()*. Toto rozhraní není tak paměťově náročné a výborně se hodí pro využití ve vestavěných zařízeních. Většina Android zařízení podporuje OpenGL ES jak ve verzi 1.0/1.1, tak ve verzi 2.0. Novější verze nabízí lepší výkon, ale za cenu komplikovanější implementace. Pro vytváření a manipulaci s grafikou pomocí OpenGL ES existují dvě základní komponenty, třída `GLSurfaceView` a rozhraní `GLSurfaceView.Renderer`.

6.2.3 GLSurfaceView

Tato třída slouží ke kreslení a k manipulaci s OpenGL ES objekty. Její metody pracují jako posluchače na události zaznamenávající pohyb prstu po displeji nebo stisknutí některého z tlačítek telefonu. Zmíněné posluchače lze nastavit pouze při zdědění třídy a přepsáním zvolených metod. Existují dva režimy vykreslování, RENDERMODE_CONTINOUSLY, kdy



Obrázek 6.1: Hlavní aktivita aplikace 3D Kompas

probíhá překreslování stále znovu a znovu nebo RENDERMODE_WHEN_DIRTY, kdy dojde k překreslení pouze při zneplatnění obsahu. V poslední řadě je zapotřebí přiřadit vykreslovací algoritmus pomocí metody *setRenderer()*.

6.2.4 GLSurfaceView.Renderer

Renderer slouží k vykreslování grafiky na určené GLSurfaceView ploše. Pro použití tohoto rozhraní jsem implementoval následující metody:

- onSurfaceCreated() – Volána pouze jednou při vytváření plochy. Zde je nutno vykonat inicializaci a konfiguraci OpenGL prostředí a objektů.
- onSurfaceChanged() – Tato metoda je volána při změně geometrie vykreslovací plochy nebo při příležitosti změny orientace displeje. Hlavním účelem je tedy opětovná inicializace rozhraní, tentokrát s jinými parametry.
- onDrawFrame() – Hlavní vykreslovací jádro celé aplikace. Metoda je zodpovědná za vykreslování aktuálních snímků.

6.2.5 (Re)Inicializace OpenGL rozhraní

Jak již bylo řečeno, inicializace se provádí v metodě *onSurfaceCreated()*. V prvé řadě jsem nastavil základní barvu, která je zobrazena, pokud není žádná barva specifikována v bufferu barev. Barvu lze nastavit pomocí metody *glClearColor()* [6]. Podobný princip

jako určení základní barvy má také určení základní hloubky pixelů pomocí metody *glClearDepth()*. V tomto okamžiku je nutné zmínit tzv. Depth Buffer, také nazývaný Z-Buffer. Ten určuje, v jaké vzdálenosti je předmět umístěn od polohy kamery, tedy umístění na ose `Z`. Nastavená hodnota se opět projeví, pokud není v Z-Bufferu nahrazena hodnotou jinou. V dalším kroku je potřeba samotný Z-Buffer aktivovat voláním metody *glEnable* s parametrem `GL10.GL_DEPTH_TEST`.

K reinicializaci dochází při změně parametrů plochy, na kterou je vykreslováno. Tím může být například změna orientace displeje telefonu. Tato situace vyžaduje znovu-určení rozměrů plochy. Metoda, která zajišťuje tuto funkcionalitu, se nazývá *glViewport()*. Protože aplikace 3D Kompas využívá celou plochu displeje, jsou funkci předány souřadnice zahrnující celý displej. Častokrát je potřeba vykreslovat pouze na určitou část displeje a právě pomocí zmíněné metody lze vybrat pouze požadovaný kus plochy a kreslit pouze na ni.

OpenGL rozhraní si po celou dobu běhu aplikace udržuje několik konkrétních matic, kde každá má specifický účel. Přístup k určité matici lze vykonat změnou tzv. módu a to metodou *glMatrixMode()*. Další částí reinicializace rozhraní je nastavení projekce, které se provádí právě při specifickém módu nazývaném `GL_PROJECTION`. Před samotnou úpravou projekce metodou *gluPerspective()* jsem "resetoval" příslušnou matici voláním *loadIdentityMatrix*. Výběr matice k resetování se provádí právě podle aktuálního módu. Nakonec je vhodné přepnutí zpět do módu `GL_MODELVIEW`, ve kterém se vykonává kreslení a další práce s modelem.

6.2.6 Vytváření modelu střelky

Model střelky tvoří 8 rovnoramenných trojúhelníků, spojených do dvou zrcadlově symetrických jehlanů. S ohledem na data model definují celkem 3 buffery. První z nich uchovává souřadnice bodů jednotlivých trojúhelníků, druhý RBGA informaci o barvě každého z bodů v předchozím bufferu a poslední poskytuje souřadnice bodů pro vykreslení čar sloužících jako kostra modelu.

Vytváření bufferu z pole souřadnic bodů typu `float` demonstruje kód 6.2.

```
ByteBuffer bb = ByteBuffer.allocateDirect(vertices.length * 4);
lbb.order(ByteOrder.nativeOrder()); // nativni razeni podle zarizeni

vertexBuffer = lbb.asFloatBuffer(); // buffer jako float
vertexBuffer.put(vertices); // vlozeni pole s polozkami
vertexBuffer.position(0); // nastaveni prvni polozky ke cteni
```

Kód 6.2: Vytváření bufferu pro OpenGL rozhraní

6.2.7 Vykreslování modelu

Vykreslování modelu probíhá ve dvou fázích. V první fázi (metoda *onDrawFrame()* ve vykreslovacím jádře) dochází k vymazání obsahu plochy a upravení pohledu a orientace scény. Po vyresetování `ModelView` matice jsem scénu posunul v záporném směru po ose `Z`, aby model nebyl přímo ve středu kamery a následně natočil podle aktuálních dat ze senzorů. Rotace se provádí pomocí metody *glRotatef()* postupně pro všechny 3 osy souřadnicového systému. Data senzorů jsou vždy aktuální, protože je hlavní aktivita doručuje průběžně. Nyní dochází k vykreslování samotného modelu. Aby OpenGL jádro vědělo, co vše má vykreslovat a na co brát ohled, specifikují se tzv. client-side capabilities, volně přeloženo klientské parametry či schopnosti. K vykreslování střelky se používá buffer s vrcholy. Provedl

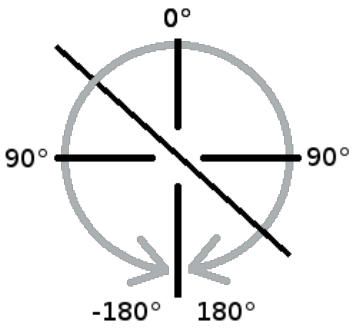
jsem tedy aktivaci parametru `GL_VERTEX_ARRAY` a metodě *glVertexPointer()* předal referenci na buffer. Obdobná situace se opakuje také pro buffer s barvami. V následujícím kroku dochází k vykreslení samotných trojúhelníků pomocí metody *glDrawArrays()*. Na závěr se sluší deaktivovat klientské parametry, které jsem před vykreslováním aktivoval.

6.2.8 Získávání a zpracování dat ze senzorů

Aplikace 3D Kompas využívá pro zjišťování orientace celkem dva senzory. Akcelerometr společně se senzorem magnetického pole. Pro větší plynulost zobrazení dochází k průměrování posledních deseti doručených souřadnic pro každou z os. Ty jsou postupně vkládány do kruhového bufferu a při vykreslování se bere v úvahu průměrná hodnota.

Transformace hodnot pro určitý rozsah

Z důvodu průměrování hodnot nastává problém s krajními hodnotami stupnic, které se diametrální liší. Stupnici znázorňuje obrázek 6.2. Pokud je telefon nakloněn v libovolné ose zhruba o 180 stupňů, hodnoty vložené do kruhového bufferu kmitají na rozhraní hodnot -180 a 180 stupňů. Při průměrování takovýchto hodnot logicky není dosaženo kýženého výsledku.



Obrázek 6.2: Způsob distribuce hodnot úhlů náklonu

Správného chování jsem docílil lokální transformací hodnot v rozsahu od 160 do 180 stupňů a od -180 do -160 stupňů, na hodnoty od 160 do 200 stupňů. Převod znázorňuje rovnice 6.1. Pokud k transformaci došlo, výsledná průměrná hodnota musí být opět převedena na původní rozsah. Zpětný převod demonstruje vzorec 6.2.

$$\text{nový úhel} = (\text{původní úhel} + 360) \mod 360 \tag{6.1}$$

$$\text{transformovaný úhel} = \begin{cases} \text{úhel}, & \text{úhel} < 180 \\ \text{úhel} - 360, & \text{úhel} > 180 \end{cases} \tag{6.2}$$

6.3 Aplikace Tracker

Aplikace Tracker slouží k záznamu absolvované trasy. Záznam probíhá volitelně s nebo bez užití GPS. Naměřená data jsou vložena do SQLite databáze. Všechny změřené trasy lze formou ListActivity zobrazit v seznamu, načež lze vybranou zobrazit přímo na mapě. K zobrazení trasy jsem využil Google Maps API. Zobrazit trasu lze tedy pouze s funkčním

internetovým připojením. Zdrojové kódy aplikace jsou umístěny na přiloženém CD v adresáří *src/Tracker*. V příloze D uvádím postup pro kompilaci projektu a jednoduchý návod na použití aplikace.

6.3.1 Základní komponenty aplikace

Aplikace se skládá ze tří hlavních aktivit. Aktivita s rozhraním pro spuštění/ukončení záznamu trasy, aktivita se seznamem všech tras a aktivita pro zobrazení trasy na mapě. Záznam trasy probíhá, ikdyž aplikace není zrovna spuštěna na popředí nebo pokud telefon spí. Tato funkcionalita je implementována pomocí služby běžící na pozadí (viz 3.3.3). Pro práci a přístup k databázi slouží třídy `DatabaseHelper` a `DatabaseAccess`.

6.3.2 Vytváření SQLite databáze

Pro prvotní vytvoření, průběžné aktualizace schématu a konečné smazání databáze jsem implementoval třídu `DatabaseHelper`. Tyto základní operace s databází probíhají formou SQL dotazů prostřednictvím metody *execSQL()*. Schéma tabulky "locations", hlavní tabulky aplikace, znázorňuje tabulka 6.1. Třída `DatabaseHelper` tvoří pomyslnou první vrstvu pro komunikaci mezi databází a aplikací. Pod druhou vrstvou, mezi třídou `DatabaseHelper` a aplikací, si lze představit třídu `DatabaseAccess`.

Sloupec	**Typ**	**Popis**
id	int	Jednoznačný identifikátor záznamu.
latitude	real	Zeměpisná šířka pořízené polohy.
longtitude	real	Zemepisná délka pořízené polohy.
time	integer	Datum a čas pořízení záznamu.
first	integer	Indikátor, jestli se jedná o první prvek zaznamenávané trasy.
collection	integer	Číslo určující, do jaké trasy bod patří.

Tabulka 6.1: Schéma tabulky "locations" databáze aplikace Tracker

Třída DatabaseAccess

Pomocí metod této třídy lze provádět vybrané dotazy a příkazy nad databází. Přístup k databázi samotné je získán přes instanci třídy `DatabaseHelper`. Kód 6.3 demonstruje získání instance třídy typu `Cursor`, který obsahuje seznam všech zaznamenaných tras. Stěžejní funkcí tohoto algoritmu je metoda databáze nazývaná *query()*. Ta slouží jako prostředek k vykonávání "SELECT" dotazů, nikoli však formou přímého SQL dotazu. Pomocí parametrů jsou metodě předány údaje jako projekce (výběr požadovaných sloupců), selekce (výběr řádků, které splňují požadovanou podmínku) aj. SQL příkaz je následně vygenerován interně.

```
public Cursor getListOfCollections()
{
  Cursor cursor = database.query(DatabaseHelper.TABLE_LOCATIONS, allColumns,
        DatabaseHelper.COLUMN_FIRST_ITEM + "=1", null, null, null, null);
  cursor.moveToFirst(); // previnuti cursoru na zacatek
  return cursor;
}
```

Kód 6.3: Získání seznamu všech zaznamenaných tras

Přístup k databázi

Z důvodu potřeby snadného přístupu k databázi je základní třída `android.app.Application` nahrazena třídou `App`, která je potomkem výše zmíněné třídy. Při vytváření samotné aplikace dochází k vytvoření statické instance třídy `DatabaseAccess`, jejíž referenci lze získat pomocí statické metody *getDatabaseAccess()*. Díky tomuto schématu lze k databázi přistupovat ze všech komponent aplikace. Při ukončení aplikace dochází k uzavření přístupu.

6.3.3 Služba zaznamenávající trasu

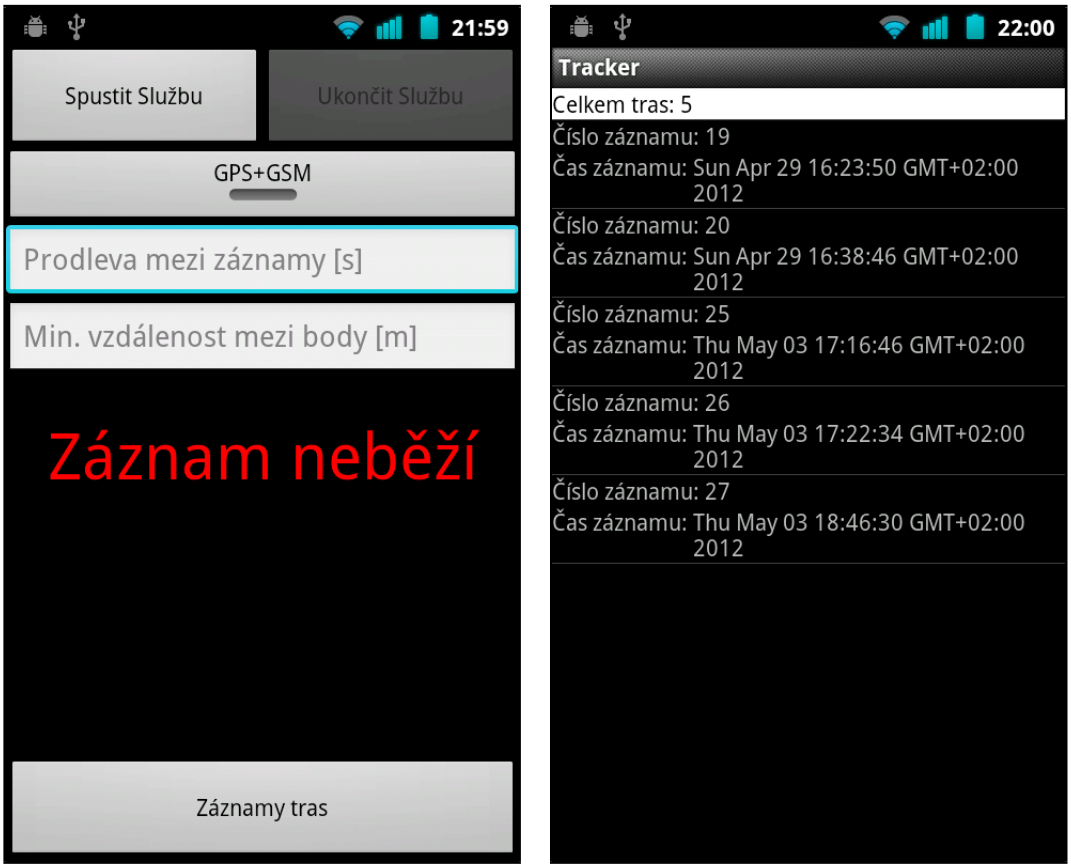
Záznam trasy probíhá ve třídě `LocationObtainService`, která je potomkem třídy `Service`. Protože služba běží ve stejném vláknu jako hlavní aktivita, případné delší výpočty či uspání by znamenalo zablokování uživatelského rozhraní a po krátké chvíli je zobrazen ARN[1] dialog. Z tohoto důvodu jsem implementoval cyklící vlákno, které je při startu služby spuštěno a běží, dokud není služba ukončena. Toto vlákno spouští další vlákno, jehož účelem je obstarání aktuální polohy. Získávání polohy probíhá pomocí GPS nebo podle údajů ze sítě. Záleží na nastavení aplikace před spuštěním služby. Vývojový diagram znázorňující start a část života služby je součástí přílohy C.

Přístup k oběma typům vláken jsem definoval jako `static`. V jednu chvíli jsou tedy se službou spjata vždy maximálně dvě vlákna. V případech, kdy se uživatel pokusil ukončit cyklící vlákno právě ve chvíli, kdy se nachází v režimu spánku, je vlákno korektně ukončeno po probuzení. Nesplňuje totiž podmínku pro další cyklení a získávání polohy. Pokud zrovna uživatel aktivuje službu znovu, vytvoří se zcela nové cyklící vlákno.

6.3.4 Seznam tras jako ListActivity

`ListActivity` je speciální případ aktivity, která slouží k zobrazování seznamu položek. Seznam položek je v aplikaci reprezentován prvkem typu `ListView`, který vyplňuje celé okno aktivity. Jeho hlavička zobrazuje počet dostupných zaznamenaných tras. Způsob přidání hlavičky demonstruje kód 6.4. Protože jsem potřeboval se seznamem spojit položky databáze složené z několika sloupců, jedinou vhodnou volbou byl `SimpleCursorAdapter`.

[1] Application Not Responding - Aplikace nereaguje



Obrázek 6.3: Hlavní aktivita aplikace Tracker (vlevo) a aktivita se seznamem tras (vpravo)

```
LayoutInflater inflater = getLayoutInflater();
// vytvoreni View z XML souboru
View v= inflater.inflate(R.layout.list_view_header, null);

// ziskani reference na TextView, ktery zobrazuje pocet tras
TextView headerText = (TextView) v.findViewById(R.id.tracks_count);
headerText.setText(String.valueOf(cursor.getCount()));

// nastaveni vytvoreneho View jako hlavicky
lv.addHeaderView(v);
```

Kód 6.4: Přiřazení hlavičky k `ListView`

Nad seznamem jsou implementovány posluchače pro krátký i dlouhý stisk. Dlouhý stisk způsobí odstranění zvolené trasy z databáze a aktualizaci seznamu. Krátký stisk spouští novou aktivitu zobrazující vybranou trasu na mapě.

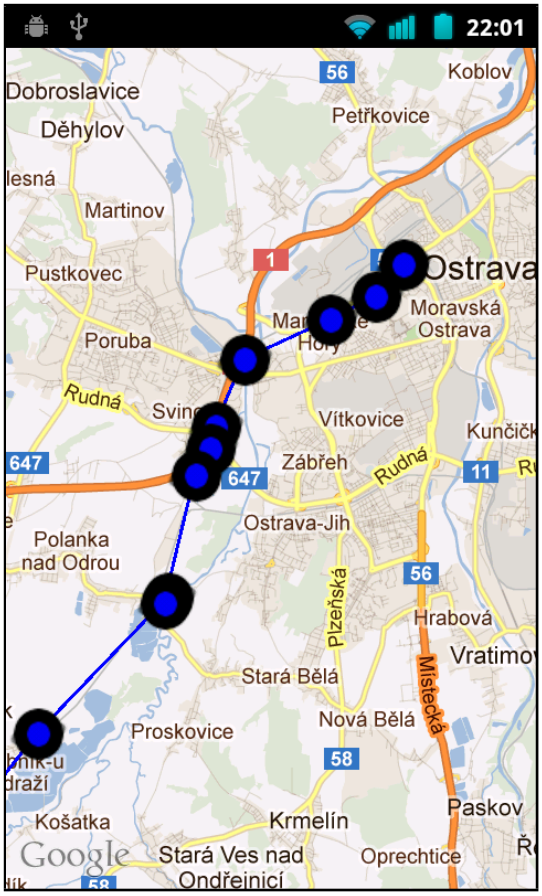
6.3.5 Google Maps API

Společnost Google poskytuje pro Android vývojáře rozhraní pro práci s Google Mapami (angl. Google Maps) [4]. Právě tyto mapy jsem použil k zobrazení zaznamenané trasy. Aktivita s mapami je typu `MapActivity`. Přes mapu jsem vytvořil dvě vrstvy.

První vrstva zobrazuje zaznamenaná místa a je potomkem třídy `ItemizedOverlay`.

U této třídy jsem přepsal metodu *onTap()*, která je volána pokaždé, když dojde ke kliknutí na některé z míst. V těle metody dojde k vytvoření dialogu s informací o zeměpisné šířce a délce, načež je dialog zobrazen.

Druhá vrstva dědí třídu `Overlay` a zobrazuje spojnice mezi jednotlivými body trasy. K samotnému vykreslování slouží přepsaná metoda `onDraw()`. V metodě je cyklus, který postupně prochází sousední prvky a zaznamenává jejich spojnice s využitím instance třídy `Path`. Jejím hlavním účelem je zapouzdřování geometrických tvarů, v tomto případě pouze úseček. Nakonec se spojnice vykreslí na `Canvas`, který je dostupný jako jeden z parametrů metody `onDraw()`.



Obrázek 6.4: Trasa zobrazená na mapě v aplikaci Tracker

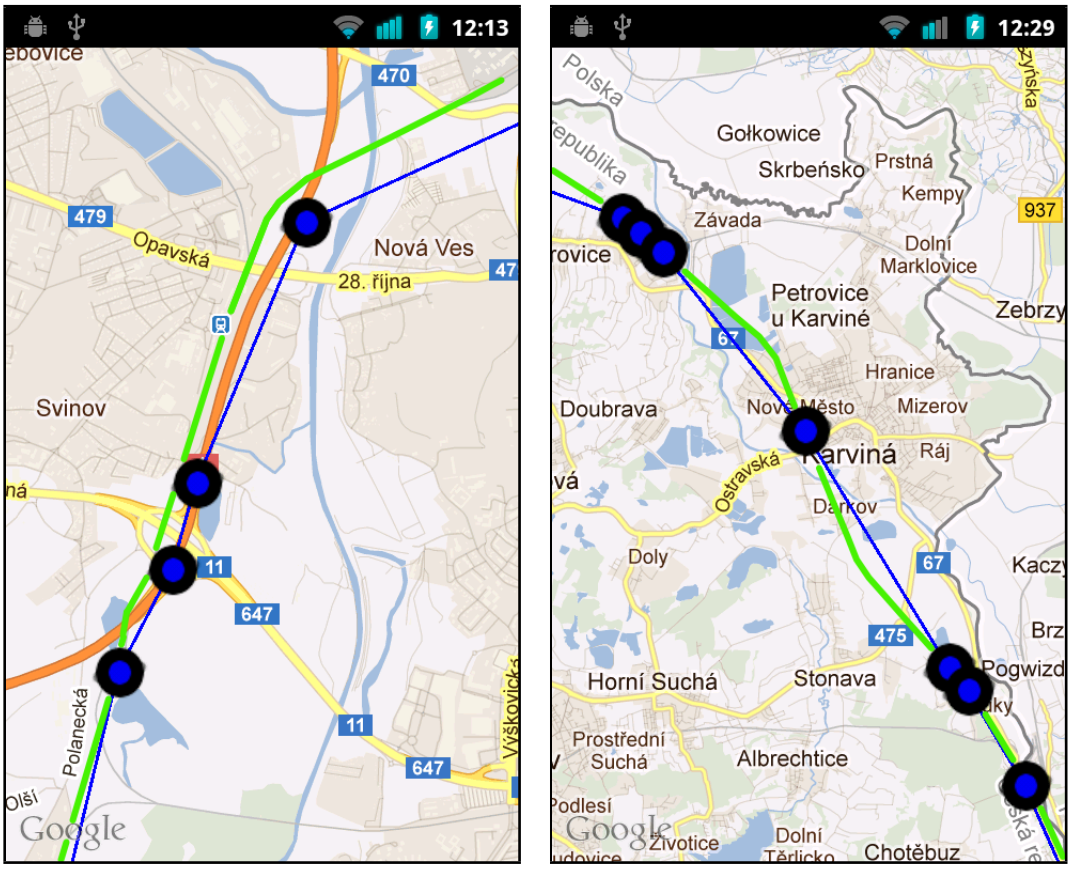
6.3.6 Rozdíly mezi záznamy s/bez GPS

Demonstrativně jsem pořídil několik záznamů, ať už pomocí GPS či pouze pomocí sítě. Záznam probíhal v jedoucím vlaku. Oba tyto přístupy znázorňuje obrázek 6.5. Skutečná trasa je vyznačena zeleně.

V případě používání GPS je zaznamenaná poloha téměř shodná s aktuální polohou, ovšem za cenu delšího běhu lokalizačního vlákna. To způsobuje větší spotřebu energie, protože zaměření polohy, tzv. *GPS fix*, trvá až několik desítek sekund.

Oproti tomu získávání polohy pomocí sítě je velice rychlé a úsporné. Poloha je získána téměř okamžitě. Tento způsob zaměřování však trpí velmi malou přesností měřených dat. V

místech, se slabým pokrytím mobilním signálem, byly výkyvy až v řádu několika kilometrů. V hustěji obydlených oblastech, které jsou zpravidla pokryty mnohem lépe, byla zjištěná poloha mnohem přesnější, v řádu desítek metrů.



Obrázek 6.5: Výsledky záznamu pomocí mobilní sítě (vlevo) a pomocí GPS (vpravo)

Kapitola 7

Závěr

Práce se zabývá principy tvorby aplikací pro operační systém Android. Postupně jsem se věnoval všem důležitým oblastem důležitých pro vývoj.

V první části jsem čtenáře seznámil se systémem Android obecně. Zmiňuji jeho architekturu a historii. Následuje popis nástrojů potřebných pro vývoj. Jedná se o editor Eclipse a balík knihoven a aplikací zvaný Android SDK. V navazujících podkapitolách jsem probral základní komponenty každé aplikace. Zabývám se způsoby ukládání dat, účelem Android Manifestu a použitím zdrojů, tzv. resources.

Aplikace samozřejmě musí disponovat grafickým uživatelským rozhraním. Tomu se věnuji v další kapitole, ve které jsem popsal základní prvky rozhraní a způsoby rozvržení těchto prvků na displeji.

Po nutném úvodu jsem popsal rozhraní pro práci se senzory. Uvedl jsem několik důležitých tříd a nastínil možnosti jejich užití.

Hlavním cílem práce bylo implementovat několik vzorových aplikací využívajících senzory. Z tohoto důvodu jsem vytvořil celkem tři aplikace, které pro svoji činnost právě senzory využívají. První aplikace, zvaná Orientace a Zrychlení, získává data akcelerometru a senzoru magnetického pole pro určení orientace přístroje. Pro určení zrychlení užívá navíc lineární akcelerometr.

Další implementovaná aplikace je 3D Kompas. Střelka kompasu se naklání v závislosti na aktuálním náklonu telefonu. Je vykreslována pomocí Open GL ES rozhraní, které slouží k vytváření a zobrazování grafických prvků.

Poslední aplikace se nazývá Tracker. Tato aplikace slouží k zaznamenávání absolvované trasy. Uživatel určí, zda chce pro získání záznamu použít GPS nebo pouze informace ze sítě. Dále specifikuje minimální vzdálenost mezi zaznamenávanými body trasy a prodlevu mezi jednotlivými pokusy o získání trasy. Záznam probíhá i v případě, že je telefon v režimu spánku. Tuto funkčnost zajišťuje služba běžící na pozadí.

V dalším vývoji by si určitě minimálně aplikace Tracker zasloužila přívětivější grafické uživatelské rozhraní. Zajímavé by také mohlo být zobrazení grafů s převýšením na absolvované trase, průměrnou rychlostí jednotlivých úseků aj.

Literatura

[1] Activity. *Android Developers* [online]. 2012 [cit. 7. 5. 2012]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html

[2] Application Fundamentals. *Android Developers* [online]. 2012 [cit. 7. 4. 2012]. Dostupné z: http://developer.android.com/guide/topics/fundamentals.html

[3] Dialogs. *Android Developers* [online]. 2012 [cit. 7. 4. 2012]. Dostupné z: http://developer.android.com/guide/topics/ui/dialogs.html

[4] Google Maps API. *Google Developers* [online]. 2012 [cit. 1. 5. 2012]. Dostupné z: https://developers.google.com/maps/documentation/android/

[5] HASHIMI, Sayed, Satya KOMATINENI a Dave MACLEAN. *Pro Android 2*. New York: Apress, 2010, 718 s. ISBN 978-143-0226-604.

[6] JONES, Tim. SDL + OpenGL Tutorial Basics. *SDLTutorials* [online]. 2010 [cit. 18. 4. 2012]. Dostupné z: http://www.sdltutorials.com/sdl-opengl-tutorial-basics

[7] MCDERMOTT, Peter. Porting Android to a new device. *LinuxForDevices.com* [online]. 2008 [cit. 9. 12. 2011]. Dostupné z: http://bit.ly/1Shyuu

[8] Motion Sensors. *Android Developers* [online]. 2012 [cit. 19. 4. 2012]. Dostupné z: http://developer.android.com/guide/topics/sensors/sensors_motion.html

[9] O'BRIAN, Terrence. Andy Rubin: over 500,000 Android activations a day, and growing. *Engadget* [online]. 2011 [cit. 17. 11. 2011]. Dostupné z: http://engt.co/mpZjvn

[10] Open Handset Alliance - Members. *Open Handset Alliance* [online]. 2007 [cit. 17. 11. 2011]. Dostupné z: http://www.openhandsetalliance.com/oha_members.html

[11] Platform Versions. *Android Developers* [online]. 2011 [cit. 9. 12. 2011]. Dostupné z: http://developer.android.com/resources/dashboard/platform-versions.html

[12] SensorEvent. *Android Developers* [online]. 2012 [cit. 5. 5. 2012]. Dostupné z: http://goo.gl/o8k31

[13] SHARKEY, Jeff. Introducing home screen widgets and the AppWidget framework. *Android Developers* [online]. 2012 [cit. 17. 3. 2012]. Dostupné z: http://goo.gl/Wg6kz

[14] SMRČEK, Jakub. Google Android – velký výlet do historie. *Cnews* [online]. 2011 [cit. 17. 11. 2011].
Dostupné z: http://www.cnews.cz/google-android-velky-vylet-do-historie

[15] Supporting Multiple Screens. *Android Developers* [online]. 2012 [cit. 17. 3. 2012].
Dostupné z: http://goo.gl/KlYz9

[16] Intents and Intent Filters. *Android Developers* [online]. 2011 [cit. 17. 3. 2012].
Dostupné z:
http://developer.android.com/guide/topics/intents/intents-filters.html

[17] What is Android? *Android Developers* [online]. 2011 [cit. 17. 11. 2011]. Dostupné z:
http://developer.android.com/guide/basics/what-is-android.html

Seznam příloh

Příloha A

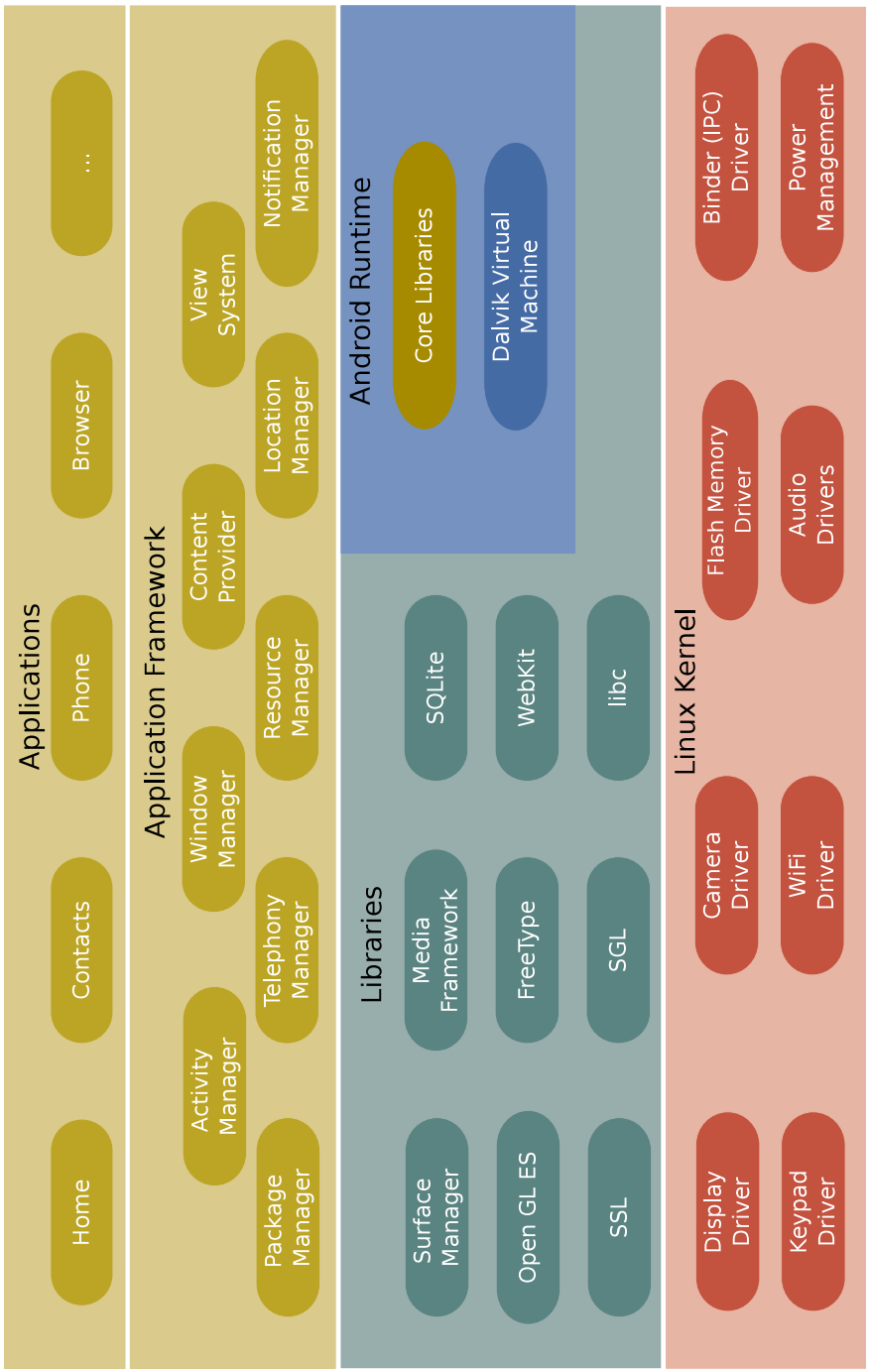
Obsah CD

Na přiloženém CD se nacházejí následující adresáře:

- `/src` – zdrojové kódy implementovaných aplikací
- `/apk` – zkompilované aplikace ze složky `/src`
- `/bp_pdf` – bakalářská práce ve formátu PDF
- `/bp_latex` – zdrojové kódy bakalářské práce v LaTeXu

Příloha B

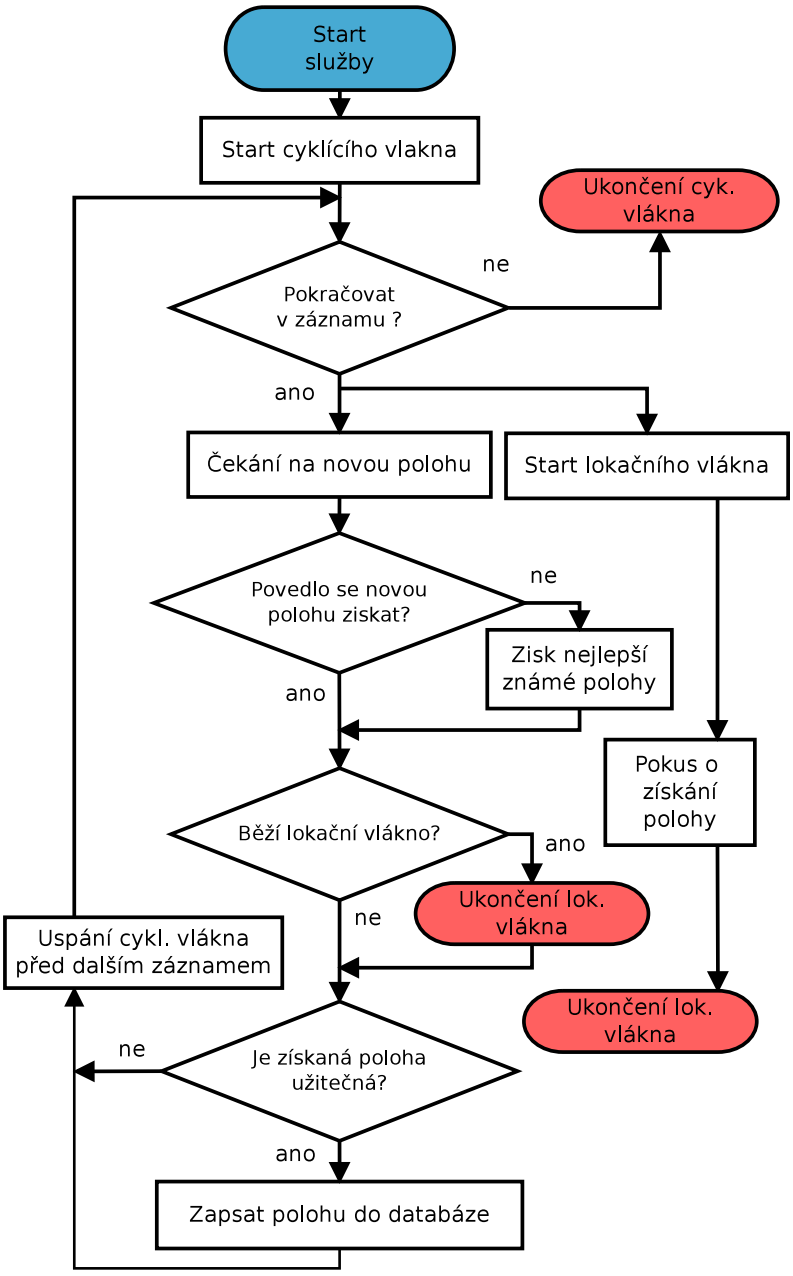
Architektura systému Android OS



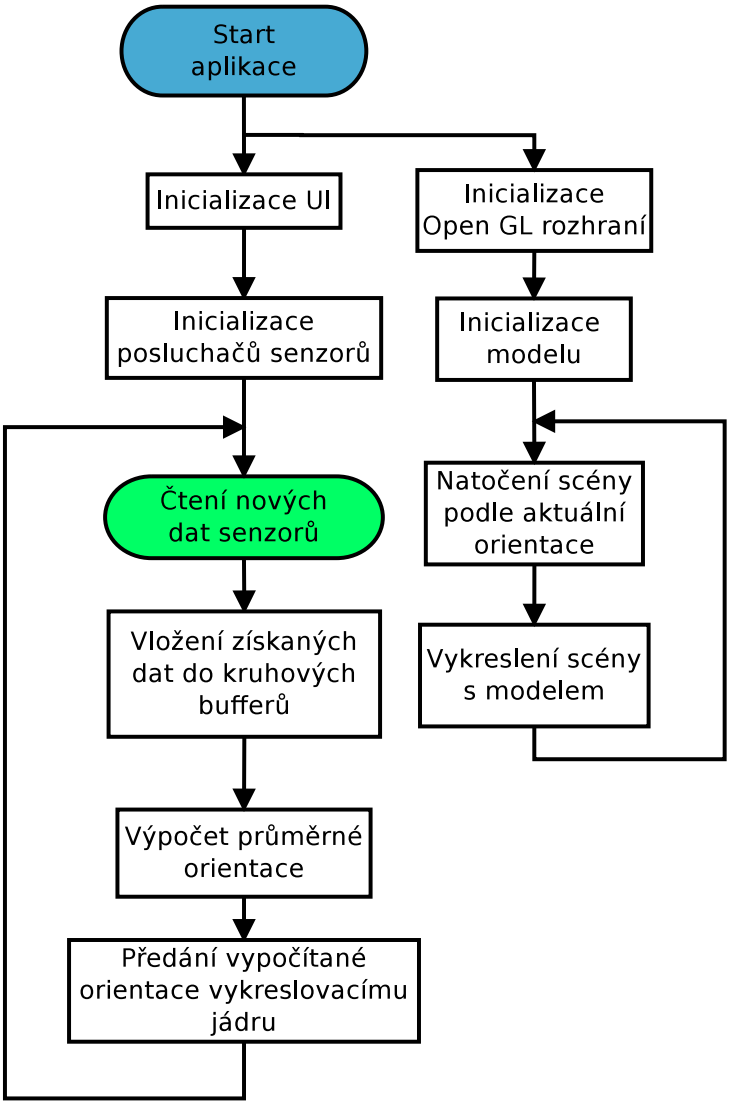
Přehled komponent Android architektury [17]

Příloha C

Vývojové diagramy aplikací Tracker a 3D Kompas



Vývojový diagram znázorňující počátek života služby aplikace Tracker



Vývojový diagram aplikace Kompas

Příloha D

Překlad a použití aplikace Tracker

Postup kompilace aplikace Tracker:

1. Stažení Android SDK[1] a Eclipse Classic[2].
2. Instalace ADT rozšíření[3] do Eclipse.
3. Import projektu do Eclipse pomocí následující posloupnosti: `File -> Import -> General -> Existing Projects into Workspace`
4. V položce "Select root directory" vybrat umístění adresáře s projektem.
5. Tlačítko `Finish` importuje projekt.
6. Nyní je nutné vygenerovat Google Maps API klíč. V opačném případě nebudou fungovat mapové podklady.
7. Klíč si může vygenerovat každý, kdo vlastní Google účet. Generátor je dostupný na: https://developers.google.com/android/maps-api-signup?hl=cs.
8. Vygenerovaný klíč je nutné vložit do souboru *res/layout/map_view.xml* do atributu `android:apiKey`.

Použití aplikace Tracker:

1. Po spuštění aplikace je potřeba určit, zdali se má provádět záznam pouze pomocí GSM sítí nebo pomocí GPS a zároveň GSM.
2. Dále je potřeba specifikovat prodlevu mezi jednotlivými záznamy a minimální vzdálenost mezi body trasy. Menší prodleva bude mít za následek větší spotřebu energie.
3. Služba se spouští tlačítkem `Spustit službu`.
4. V tuto chvíli je možné telefon uspat, záznam přesto poběží.
5. Pro ukončení záznamu slouží tlačítko `Ukončit službu`.
6. Pořízený záznam je následně dostupný v seznamu tras. Ten lze zobrazit kliknutím na tlačítko `Záznamy tras`.
7. Klikem na požadovovanou trasu ji lze zobrazit na mapě. Dlouhým stiskem dojde k vymazání trasy z databáze.

[1] http://developer.android.com/sdk/index.html
[2] http://www.eclipse.org/downloads/
[3] http://developer.android.com/sdk/eclipse-adt.html